INTERBUS-S

User Manual

Hardware and Firmware of the Controller Board for IBM®-compatible PCs

Type: IBS PC CB HW UM E

Revision: C

Order No.: 27 47 16 7

This manual is valid for the following controller boards with the firmware version 3.72:

IBS PC CB/I-T	Order No.: 27 80 84 9
IBS PC CB/COP/I-T	Order No.: 27 54 51 6
IBS PC CB/RTX486/I-T	Order No.: 27 61 47 0

PHOENIX CONTACT

Please Observe the Following:

In order to guarantee that your use of this manual is as straightforward as possible and that hardware is used safely in the installation, operation and maintenance phases, we request that you carefully read and observe the following instructions:

Explanation of Symbols Used

The *attention* symbol refers to erroneous handling, which could lead to damage to the hardware or software, or, in indirect connection with dangerous process peripherals (e.g., unprotected shafts or motors with actuator functions), to light to severe personal injury. The symbol is always located to the left of the tagged text.

The *hand* symbol gives you tips and advice on the efficient use of hardware and on software optimization, to save you from performing extra work, for example. In addition, text marked in this way informs you of system-related maximum and minimum conditions that must absolutely be observed to achieve error-free operation. The hand is also found in front of clarifications of terms.

The *text* symbol refers to detailed sources of information (manuals, data sheets, literature, etc.) on the subject matter, product, etc. This text also provides helpful infomation for the orientation, reading order, etc. in the manual.

We are Interested in Your Opinion

We are constantly attempting to improve the quality of our manuals. Should you have any suggestions or recommendations for improvement of the contents and layout of our manuals, we would appreciate it if you would send us your comments. Please use the universal telefax form at the end of the manual for this.

Statement of Legal Authority

Phoenix Contact reserves the right to make any technical changes that serve for the purpose of technical progress.

Phoenix Contact reserves all rights in the case of a patent award or listing of a registered design. External products are always named without reference to patent rights. The existence of such rights shall not be excluded, however.

The use of products described in this manual is oriented exclusively to qualified application programmers and software engineers familiar with automation technology and the applicable national standards. Phoenix Contact assumes no liability for erroneous handling of or damage to Phoenix Contact or external products resulting from disregard of information contained in this manual.

Table of Contents

Section 1

Introduction and Overview

This section provides:

- a short introduction into controller board parameterization and programming;
- a short overview of the documentation available for InterBus-S.

1 Introduction and Overview

The IBS PC CB/.../I-T controller boards are used to connect InterBus-S to IBM-compatible PCs. The documentation is to provide information on all functionalities and applications. This requires a document size which does not always make it easy to locate the required information within a short time.

Therefore, we want to give you here a short overview of the controller board handling. The individual steps are described in detail in the following chapters. Should you still have questions after studying the manual, call our technical hotline under the phone number Germany: 5235 / 34 18 88.

1.1 Quick Start Under DOS

The controller board requires 8 bytes in the I/O area of your PC, a free interrupt and 4 kbytes of free address space. The default settings are so that they often need not be changed.

1. Ascertain whether the default settings is right for your PC configuration:

I/O address	120_{hex}
Interrupt	15
Memory address	$D000_{hex}$

Always make sure that the settings selected for the controller board are not used by other components of your PC. For example, the memory area between 640 kbytes and 1 Mbyte is often used by drivers, etc. Select other values if necessary (see Section 4). Double assignments are the most frequent sources of startup errors!

2. Remove power from your PC (switch off and remove power cable) and install the controller board. Be sure that the controller board does not rest on or contact other components in the PC (short-circuit hazard)!
3. Switch on the PC again. The easiest controller board startup method under DOS is to use the *PCCBMONI.EXE* monitor program supplied on the tool diskette (see Section 4).

   Before starting the monitor program, be sure to start the driver *IBSPCCB.EXE* for the controller board. If you retained the default settings for your controller board, you do not need to transfer parameters when calling the driver. The batch file *MONI.BAT*, which is also on the diskette, provides an easy way of starting the drivers and the monitor program with one call. When you have connected a bus configuration, you can check it with the monitor program without the need to program a single line.

1.2 Programming - Fundamentals

The driver software accesses the controller board's multi-port memory via a 4 kbyte window in the PC memory area between 640 KB and 1 MB. The following functions are available for this:

- Functions for opening and closing data channels
- Functions for writing commands and reading messages (mailbox interface, works with handshake signals and interrupt control features)
- Functions for reading and writing I/O data (data interface, works without acknowledgment)
- Diagnostic functions for monitoring the controller board's state

When opening a data channel, you will receive a *node handle* in response, which, similar to the *handle* in the case of a file access, specifies the data channel and must be entered when data is to be read or written (see driver software manual *IBS PC CB SWD UM E*, Order No.: 27 53 96 0).

InterBus-S control

Your application program starts the controller board by means of commands (e.g. start of the bus system, reading in the bus configuration, see Section 8).

After its initialization and the start of the data transmission, the controller board operates the bus independently and returns messages (see Section 9).

Section 6 describes the InterBus-specific programming such as
- the physical addressing of the IBS devices,
- the logical addressing of the IBS devices, and
- the combination of the IBS devices into groups

on the basis of a configuration example.

In the event of a serious error (e.g. interrupted bus cable), all connected IBS devices automatically go into the RESET state and set their outputs to zero. Thereupon the controller board examines the error and gives a detailed description of the error cause and the error location using the *Bus_Error_Information_Indication* error message ($80C4_{hex}$, see Section 9).

Program examples

Controller board programming under DOS, Microsoft Windows® and IBM OS/2® is mainly identical. Thus, programming under DOS can be based without problems on a DOS program example. The driver diskette provides program examples, which already contain the required bus handling. You can fully concentrate on programming your I/O links.

Training

You will have found that it takes a rather long time to become familiar with all features of extensive software packages such as *Microsoft Winword®* or *Excel®*.

The same is true for InterBus-S. Of course you can familiarize yourself easily with the controller board programming using the documentation and the supplied example programs. To be able to take full advantage of all their features, we recommend in addition to attend one of our programming training courses, where you can acquire an extensive practical knowledge. For the contents and dates please refer to our seminar brochure, which your local Phoenix representative will be pleased to send you. On request, we can also hold a training course, tailored to your particular requirements, on your premises. Please contact us!

1.3 Documentation

The following documentation is available for the controller boards

- IBS PC CB/I-T,
- IBS PC CB/COP/I-T und
- IBS PC CB/RTX486/I-T:

Description of the hardware

This manual (IBS PC CB HW UM E) describes the hardware of the three controller boards and the InterBus-specific programming. In addition, it includes a librarary with commands and messages for the IBS master board (firmware version 3.72).

Description of the driver software

The *IBS PC CB SWD UM E* manual (Order No. 27 53 96 0) describes driver software for the operating systems *DOS*, *Microsoft Windows®* and *IBM OS/2®* in connection with various compilers. It is also included in the driver software package *IBS PC CB SWD* (Order No. 27 64 70 7)

Description of the developing environment TDOS-PRO

(only for IBS PC CB/COP/I-T)

The developing environment *TDOS-PRO* (IBS PC COP SWT, Order No. 27 52 12 3*)* allows to download a program to the *COP386*. The relevant documentation is enclosed.

Description of the operating system RTXDOS

(only for IBS PC CB/RTX486/I-T)

The RTX-DOS manual describes the special extensions and features of the DOS-compatible operating system for the *COP486*.

Communication via InterBus-S (PCP)

The **P**eripherals **C**ommunication **P**rotocol (PCP) is used for transmitting parameterization data to intelligent IBS devices or for communicating with an IBS device with V24 interface. PCP is a software interface based on the InterBus-S basic protocol and allows the transmission of non-time-critical large volumes of data almost independent of the process data.

The *IBS PCP UM E* manual (Order No. 27 53 93 1) describes the fundamentals and the application of the Peripherals Communication Protocol.

Configuring your InterBus-S system

The configuration manual *IBS SYS PRO UM E* (Order No. 27 51 00 1) provides information on the selection of components for your IBS system. It describes the following features:

- Electrical characteristics (voltage and current ranges)
- Mechanical characteristics (degree of protection, type of connection, installation possibilities, etc.)
- Program characteristics (e.g. required address area in the host)

The manual introduces the design types of the Phoenix Contact product families available, and their features and ordering data.

In addition, a collection of all data sheets available is enclosed.

Installing your InterBus-S system

Please refer to the installation manual IBS SYS INST UM E (Order No. 27 54 80 4) for installation instructions for I/O components (IBS devices, modules, cables etc.) The manual contains several sections with information on the following subjects:

- System overview
- Installation of the I/O components
- Recommended cabling
- I/O startup and function test
- Fault clearance
- Replacement of IBS components
- Cable plans

Table 1-1: Ordering data for the documents available

Description	Type	Order No.
Controller board system folder: - Controller board user manual - Tool diskette - Manual on the fundamentals and application of the Peripherals Communication Protocol (PCP)	IBS PC CB UM E	27 54 75 2
Controller board user manual, separate copy (also included in the system binder)	IBS PC CB HW UM E	27 47 16 7
Driver software manual (is also supplied with the driver software package *IBS PC CB SWD*)	IBS PC CB SWD UM E	27 53 96 0
Manual on the fundamentals and application of the Peripherals Communication Protocol (PCP), separate copy (also included in the system binder)	IBS PCP UM E	27 53 93 1
Configuration manual for IBS systems	IBS SYS PRO UM E	27 51 00 1
Installation manual for IBS components	IBS SYS INST UM E	27 54 80 4
Data sheets on new digital and analog I/O modules	On request	

1.4 Modular Design of the Controller Board

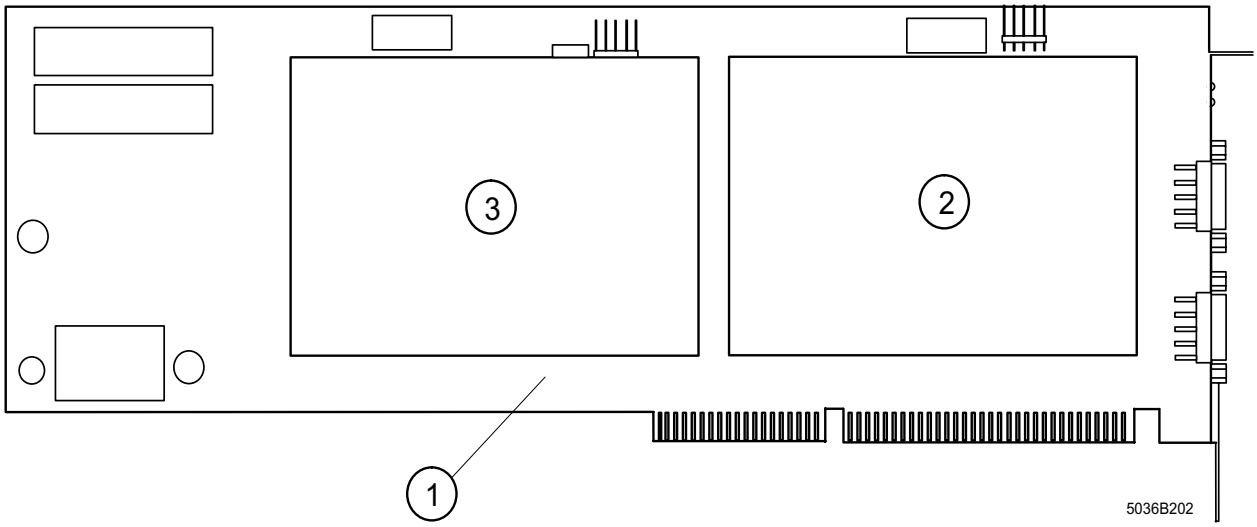


Figure 1-1: Modular design of the controller board

The controller boards IBS PC CB/I-T, IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T have modular designs and consist of the following components:

1 Motherboard
2 IBS master board (MA)
3 Coprocessor board (COP386 or COP486)

Only the controller boards IBS PC CB/ COP/I-T and IBS PC CB/RTX486/I-T have coprocessor boards. On the controller board IBS PC CB I-T it cannot be retrofitted, as the motherboard does not provide the required interfaces and the voltage supply.

The following sections describe the controller board in detail.

Section 2

Technical Description of the Motherboard

This section provides information on

- the structure and the components of the motherboard.

2 Technical Description of the Motherboard

2.1 Short Description

The IBS PC CB/.../I-T controller boards are used to interface InterBus-S to an 100% IBM-compatible standard PC (AT, 80386, 80486 etc.), which will be referred to as "host PC" in the following. The controller boards are designed as plug-in boards for long AT bus slots. The front plate is a common PC board holder.

The controller motherboard provides two slots for daughterboards. The first daughterboard is the IBS master (abbreviated: MA), which is used as an interface to InterBus-S. As a second daugtherboard, a coprocessor board (abbreviated: COP) can be used as a fast processor for InterBus-S. The front plate has two status indicator LEDs for for each daughterboard.

The central functional unit of the motherboard is a Multi-Port Memory (MPM) with an integrated access management. This memory can be accessed from the host PC and from the two daughterboards, i.e. the MPM has three nodes. The MPM consists of 64 Kbytes of SRAM with a data word width of 16 bits. It is used for exchanging commands, messages and data between the two daughterboards and the host PC. In addition, the motherboard contains the MPM access control mechanisms.

Up to four controller boards can be operated in a host PC. As different board numbers are set (from 1 to 4) while the I/O address remains the same, the controller boards differ by an I/O address offset of 0_{hex}, 8_{hex}, 10_{hex} and 18_{hex}, which depends on the board number.

A controller board occupies 8 bytes in the I/O address area, and a memory window of 4 Kbytes in the memory area of the host PC. The base address of the I/O address area can be set with a DIP switch to one of 16 possible addresses. The base address of the 4Kbyte memory window to the MPM can be set with the driver software via an I/O address.

A voltage monitoring circuit ensures the operating reliability of the controller board. In addition to an extensive reset system it is possible to reset each individual daughterboard by a specific software command. An additional function for reliability enhancement consists of watchdogs monitoring the coprocessor board and, via the AT bus, the host PC.

For the interrupt-driven operation of the controller board, the MPM control logic activates specific interrupt signals for each node. The user can adapt the interrupt system to the configuration of the host PC.

The motherboard of the IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T contains also a battery pack (6V) for backing up the SRAM and the real-time clock on the coprocessor board.

2.2 Mechanical Design

The motherboard of the IBS PC CB/.../I-T series is designed as a PC board with the standards dimensions of 338.5 mm * 114.3 mm (13.33 in. * 4.5 in.). There are two PC board-style edge connectors at the lower edge of the motherboard for the connection to the AT bus (ISA) of your PC.

2.2.1 Layout of the Controller Board

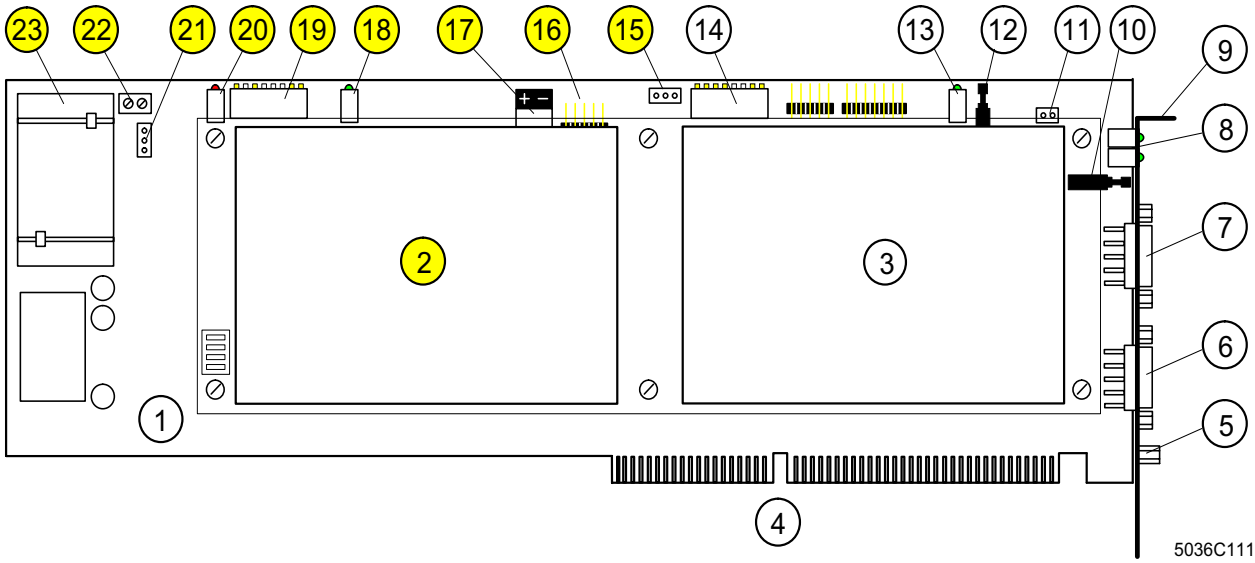


Figure 2-1: Layout of the IBS PC CB/.../I-T controller boards

The controller boards consist of the following components (the components identified by numbers with a gray background are not available on the controller board IBS PC CB/I-T):

1 Motherboard
The motherboard accepts up to two daughterboards and is the interface to the host PC AT bus.

2 Coprocessor board
for data preprocessing and control (not IBS PC CB/I-T).

3 IBS master board
The IBS master board (MA) controls the InterBus-S data traffic.

4 AT bus edge connector
The connector connects the controller board with the host PC AT bus (ISA).

5 Ground pin
Ground the controller board with the ground pin.

6 InterBus-S
remote bus interface (two-wire)

7 Serial interface (RS-232 level) for the IBS master board, for connecting a PC with the software tools IBS CMD SWT or IBS SYS SWT.

8 Status LEDs, external
The green LEDs indicate the controller board status, see Section 2.2.2.

9 Front plate
PC board holder

10 External reset button
Concealed reset button which can be pressed while the PC housing is closed.

11 Reset button jumper
For deactivating the internal and the external reset button

12 Internal reset button
For convenient operation while the PC housing is open.

13 Internal IBS master board status LED (green)
For convenient status indication while the PC housing is open.

14 DIP switch
for setting the base address in the PC's I/O area and the board number

15 Jumper, reset (not IBS PC CB/I-T)
For separating the controller board from the PCs hardware reset. This jumper is used to configure the controller board so that it keeps operating even when a PC hardware reset has been initiated.

16 Serial interface (not IBS PC CB/I-T)
RS-232 interface of the coprocessor board

17 Connector for the RAM back-up battery pack on the coprocessor board (not IBS PC CB/I-T)

18 Internal coprocessor board status LED (green, not IBS PC CB/I-T)
For convenient status indication while the PC housing is open.

19 DIP switch (not IBS PC CB/I-T)
for setting the coprocessor board boot configuration (not IBS PC CB/I-T)

20 Coprocessor board voltage supply indicator LEDs (not IBS PC CB/I-T)
The green LED indicates that the external supply voltage for the controller board is applied.
When the red LED is constantly on, the battery pack voltage has fallen below the permissible minimum value. In this case please replace the battery pack.

21 Supply jumper (not IBS PC CB/I-T)
The jumper is used to select whether the controller board is to obtain its supply voltage from the AT bus or via the terminals for external supply.

22 Supply terminal (not IBS PC CB/I-T)
Terminal for feeding in the external supply voltage. This terminal is used to supply the controller board with a voltage (5V DC 1.5A) from an external power pack.

23 Battery pack (not IBS PC CB/I-T)
for back-up of the CMOS-SRAM and of the COP real-time clock.

Front plate layout

The front plate (PC board holder) is equipped with two 9-position subminiature D connectors for the I/O interfaces of the IBS master board. Control elements are four function LEDs and a reset button, located behind the board holder. Underneath the subminiature D connector there is a round pin with internal thread, which can be used for controller board grounding (PE). Figure 2-3 shows the locations of the components on the PC board holder.

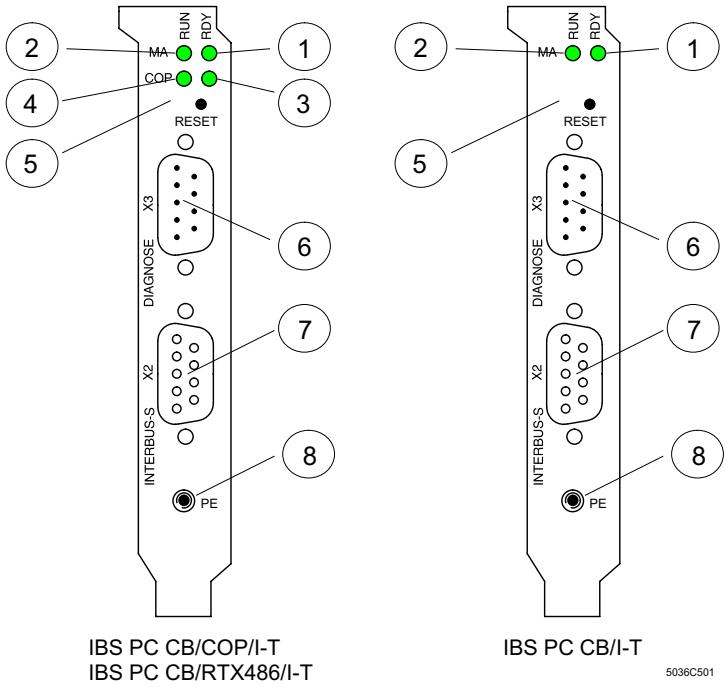


Figure 2-2: Elements on the PC board holder

1 External LED *MA READY* (master board ready) (green)
2 External LED *MA RUN* (InterBus started) (green)
3 External LED *COP READY* (coprocessor board ready) (green)
4 External LED *COP RUN* (coprocessor board active) (green)
5 External reset button
6 IBS diagnostic interface (9-position subminiature male connector, RS232 level)
7 IBS remote bus interface (9-position subminiature D female connector)
8 Ground pin

Ground the controller board with the ground pin.

2.2.2 Function of the LEDs

2 LEDs (Figure 2-2) per daughterboard are provided on the front plate (PC board holder). They indicate the *Ready* and *Run* states:

IBS master board

MA READY: After switching on, the IBS master board carried out a boot check for all functional units including MPM and is ready.

MA RUN: The IBS master board has started InterBus-S. ID or data cycles are being transmitted.

Coprocessor board

(not IBS PC CB/I-T)

COP RUN: The coprocess board operating system has booted; an application program can be started.

COP READY: A program is running on the coprocessor board.

Coprocessor board booting is followed by the automatic start of various utilities, some of which remain as TSR programs in the COP memory. Therefore, the *COP READY* LED is lit when system startup has been completed.

For convenient observation while the PC housing is open for startup, the LEDs are provided once more at the board edge opposite the AT bus.

There are two 8-way DIP switches at the top edge of the controller board for the setting of the I/O board address and the board number, and for the power-up configuration. They are easy to operate even in the built-in condition.

The settings of the DIP switches are only read in when the controller board boots. After the setting has been changed, the controller board must be reset to make the change effective.

2.3 Controller Board Interfaces

The following sections describe the various controller board interfaces.

2.3.1 Connection of the Motherboard with Your PC

The AT bus edge connectors connect the controller board with the AT bus of the host PC. The signal assignment is compatible with the ISA standard; the controller board requires only a data bus width of 8 bits.

Tables 2-1 and 2-2 show the pin assignments of the edge connectors to the AT bus. The interface signals are exclusively standardized AT bus signals.

Table 2-1: Pin assignment of the short AT bus edge connector

Pin	Signal	Function
C1		
C2	LA23	I, T, PD100
C3	LA22	I, T, PD100
C4	LA21	I, T, PD100
C5	LA20	I, T, PD100
C6		
C7		
C8		
C9		
C10		
C11		
C12		
C13		
C14		
C15		
C16		
C17		
C18		

Pin	Signal	Function
D1		
D2		
D3		
D4	IRQ10	O, C, TS
D5	IRQ11	O, C, TS
D6	IRQ12	O, C, TS
D7	IRQ15	O, C, TS
D8		
D9		
D10		
D11		
D12		
D13		
D14		
D15		
D16	+5 V	SV
D17		
D18	GND	SV

Key:

I = Input	SV = Supply voltage	TS = Tristate
O = Output	T = TTL level	PU ... = Pullup ... [kΩ]
B = Bidirectional	C = CMOS level	PD ... = Pulldown ... [kΩ]
M = For measuring only	OC = Open collector	UN ... = Nominal voltage[V]

Table 2-2: Pin assignment of the long AT bus edge connector

Pin	Signal	Function
A1		
A2	SD7	B, T, TS
A3	SD6	B, T, TS
A4	SD5	B, T, TS
A5	SD4	B, T, TS
A6	SD3	B, T, TS
A7	SD2	B, T, TS
A8	SD1	B, T, TS
A9	SD0	B, T, TS
A10	IOCHRDY	O, C, PU10
A11	AEN	I, T
A12	SA19	I, T
A13	SA18	I, T
A14	SA17	I, T
A15	SA16	I, T
A16	SA15	I, T
A17	SA14	I, T
A18	SA13	I, T
A19	SA12	I, T
A20	SA11	I, T
A21	SA10	I, T
A22	SA9	I, T
A23	SA8	I, T
A24	SA7	I, T
A25	SA6	I, T
A26	SA5	I, T
A27	SA4	I, T
A28	SA3	I, T
A29	SA2	I, T
A30	SA1	I, T
A31	SA0	I, T

Pin	Signal	Function
B1	GND	V
B2	RESET	I, T
B3	+5 V	SV
B4	IRQ2/9	O, C, T, S
B5		
B6		
B7		
B8		
B9	+12 V	SV
B10	GND	SV
B11	SMEMWL	I, T
B12	SMEMRL	I, T
B13	IOWL	I, T
B14	IORL	I, T
B15		
B16		
B17		
B18		
B19	REFRESHL	I, T
B20		
B21	IRQ7	O, C, TS
B22		
B23	IRQ5	O, C, TS
B24		
B25	IRQ3	O, C, TS
B26		
B27		
B28	BALE	I, T
B29	+5 V	SV
B30	OSC	I, T
B31	GND	SV

Key:

I = Input
O = Output
B = Bidirectional
M = For measuring only
SV = Supply voltage
T = TTL level
C = CMOS level
OC = Open collector
TS = Tristate
PU ... = Pullup ... [kΩ]
PD ... = Pulldown ... [kΩ]
UN ... = Nominal voltage[V]

2.3.2 Connection Between Motherboard and the Daughterboards

Two 58-position female connectors for the master board and two for the coprocessor board provide the connection between the motherboard and these two daughterboards. Both interfaces are assigned in the same way the address, data and control lines of the Multi-Port Memory (MPM) and slot-specific MPM signals. In addition, the signals of any I/O interfaces are supplied to the connectors of both interfaces. They are divided into the I/O signals, which are at system potential, and the electrically isolated IBS signals.

2.3.2.1 Interfaces of the IBS Master Board

The IBS master board has two interfaces, the signals of which are supplied via the 58-position female connectors to the motherboard connectors described below.

Diagnostic interface (serial)

An IBM-compatible PC with the software IBS SYS SWT or IBS CMD SWT can be connected as a diagnostic device via the diagnostic interface (RS-232 level). The diagnostic interface connector on the front plate is a 9-position male subminiature D connector.

The diagnostic cable IBS PRG CAB (Order No. 28 06 86 2) as shown below connects the diagnostic interface with the IBM-compatible PC.

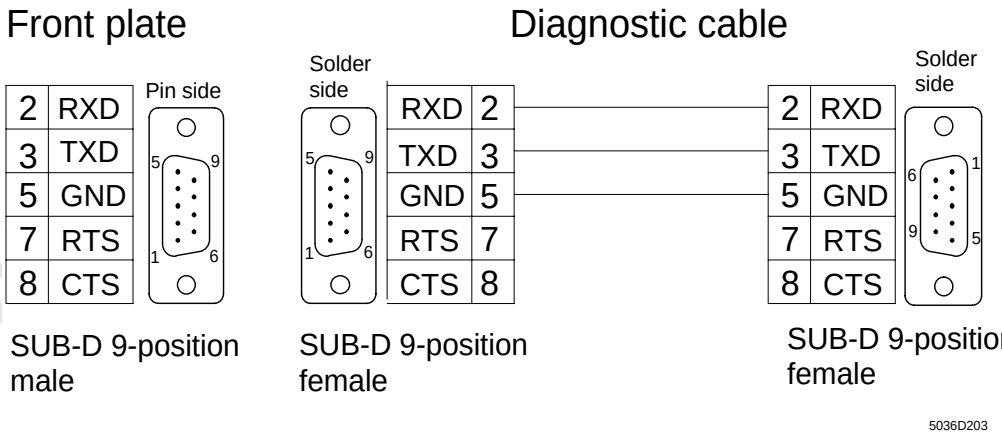


Figure 2-3: Diagnostic interface and diagnostic cable for the connection of a PC

Table 2-3: Pin assignment of the diagnostic interface

Pin	Signal	Function
1	-	Reserved
2	RXD	Received Data
3	TXD	Transmitted Data
4	-	Reserved
5	GND	Digital Ground
6	-	Reserved
7	RTS	Request to Send (not firmware-supported)
8	CTS	Clear to Send (not firmware-supported)
9	-	Reserved

Connecting the controller board with a diagnostic PC requires only the signals TXD, RXD and GND.

Remote Bus Interface

The remote bus interface is used to connect the remote bus (2-wire) of the IBS system. It is accessible on the front plate as a 9-position subminiature D female connector and electrically isolated from the host potential. The connector housing is conductively connected with the PC board holder.

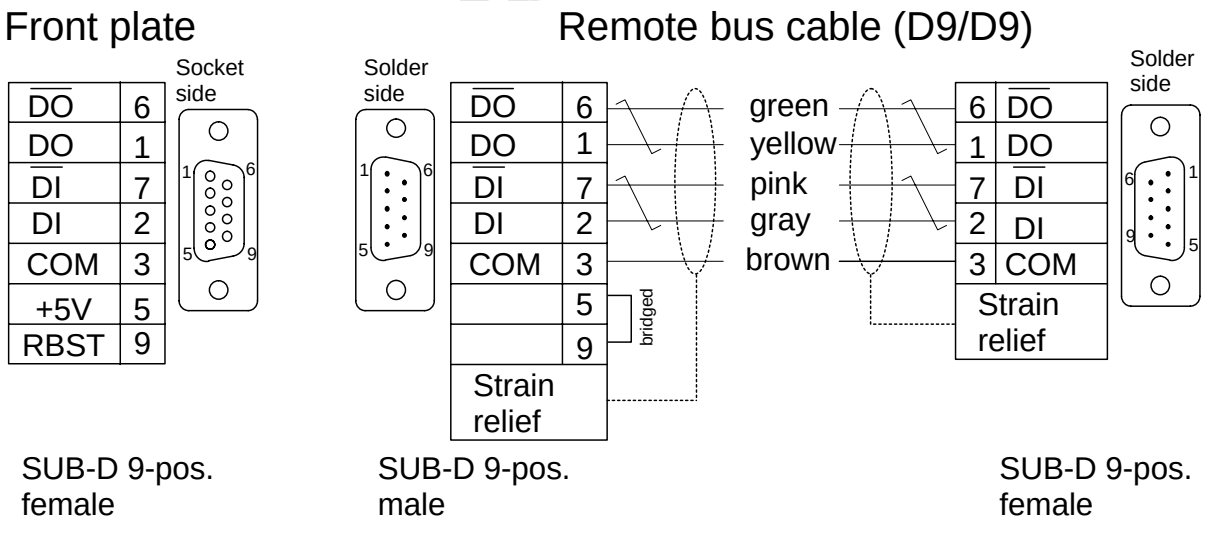


Figure 2-4: Remote bus interface and example of a remote bus cable (cable type D9/D9)

The bridge between pin 5 and pin 9 indicates to the outgoing remote bus interface of an IBS device that the outgoing remote bus cable has been connected.

Table 2-4: Remote bus interface pin assignment

Pin	Signal
1	DO
2	DI
3	COM
4	Reserved
5	+5 V
6	DO
7	DI
8	Reserved
9	RBST

For detailed information on the cable specification, all IBS cable types and the installation of your IBS system please refer to the installation manual *IBS SYS INST UM E* (Order No. 27 54 80 4).

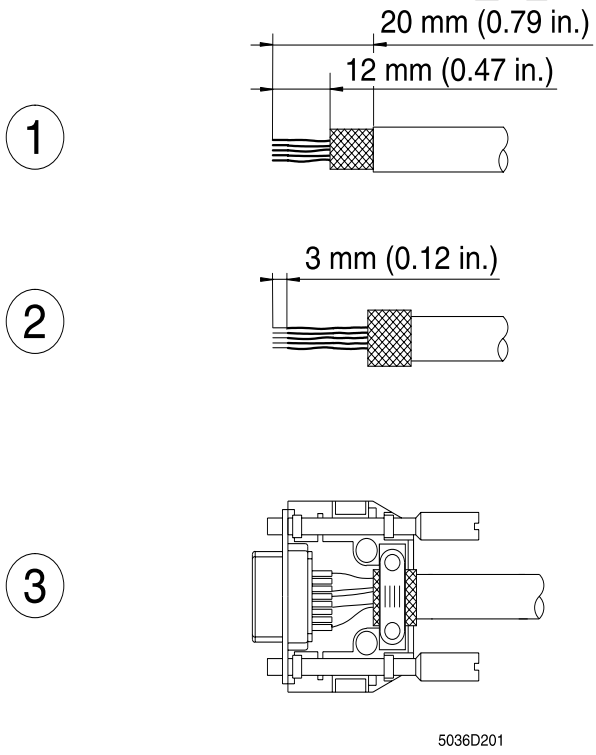


Figure 2-5: Workmanlike connection of a remote bus connector (subminiature D 9)

Connect the connector as follows:

1 Cut off 20 mm (0.79 in.) of the cable sheath and 12 mm (0.47 in.) of the shield braid.

2 Strip off 3 mm (0.12 in.) of the wire ends, and fold the shield braid evenly back onto the cable sheath.

3 After soldering on the wire ends, clamp a large surface of the shield braid,

which now lies on the cable sheath, under the conductive strain relief.

For fault-free InterBus-S operation, the remote bus cable shield braid must be connected to PE on the controller board. Use only subminiature D connectors with metal-plated or metal housings. Always ensure that a large surface of the remote bus cable shield braid is in contact with the conductive strain relief conductively connected with the connector shell. Ground the controller board via the ground pin (Figure 2-2).

2.4 Controller Board Functional Units

2.4.1 AT Bus Interface

The AT bus interface has an 8-bit data bus and a 24-bit address bus. Besides data, address and control line buffering, it comprises the address decoding feature and the setting facilities for the 8-byte I/O address area and the 4 Kbyte memory window to the MPM. In addition, interrupt signals from the MPM control logic are supplied to the AT bus for the communication with the daughterboards via the MPM. They can be switched to the interrupt inputs of the PC. The functionality of the AT bus interface is described in the following.

2.4.1.1 I/O Base Address in the Host

For each controller board, an I/O address area of 8 bytes is required for the status register, the mapping register and the control ports. Using a DIP switch, the base address of this address area can be set to one of sixteen possible addresses; it must be defined before the controller board start-up. Take care to avoid address conflicts with other boards of the host PC. The base addresses that can be set and the corresponding DIP switch setting on the motherboard are specified in Section 5.

The AT bus interface is so designed that four controller boards can share an I/O base address of the host PC. The four controller boards are then, depending on a board number (1 to 4) that can be set with a DIP switch, addressed with an offset of 0_{hex}, 8_{hex}, 10_{hex} and 18_{hex} relative to the base address. Section 5 shows the relationship between the board number and the offset to the base address that has been set, and the corresponding DIP switch setting.

2.4.2 Multi-Port-Memory

The Multi-Port-Memory (MPM) of the controller boards is used for exchanging data between the host PC and the two daughterboards. The MPM is a static RAM (SRAM) and has a storage capacity of 64 Kbytes.

Please note the following when accessing the MPM: The IBS master board uses the Motorola format (68xxx family) when placing its data in the MPM, whereas the host processor and the COP processor expect data always in the Intel format. These two formats use opposite orders of addressing the bytes in a data word. For MPM acess use the supplied macros, which exchange the high byte and the low byte when accessing the data word (see the Driver Software Manual IBS PC CB SWD UM E).

The MPM contains the mailbox interface (MXI) for commands and messages, and the data interface (DTI) for process data. In addition, several hardware registers are mapped to the MPM address area. They are used for exchanging status and handshaking information.

Each node has its own memory areas in the MPM. Every node may write to its own area, but may only read from the areas of the other nodes.

2.4.2.1 MPM Access Method

As the nodes can access the MPM completely asynchronously, the following access conflicts could occur:

- Simultaneous or overlapping reading to/writing from the same memory location,
- Reading from a logically coherent data area by a node, while this area is being written to by another node.

The first type of conflict is prevented by circuitry measures in the MPM access management. In the event of simultaneous access, the order of priorities is:

- Priority 1: host PC
- Priority 2: daughterboard 1 (IBS master)
- Priority 3: daughterboard 2 (coprocessor board)

When access operations overlap, the current access is first completed.

The second type of conflict is solved by the fact that the number of successive byte accesses per data transfer can be set variably. The MPM control logic provides data consistencies of 8, 16, 32 and 48 bits. The MPM access management inhibits MPM accesses by the other nodes until the last byte access of a data transfer has been completed.

In the event of an access conflict in the MPM, a node is stopped by a ready signal until the other node has completed its access. The access by a node is not completed before it has fetched all data bytes according to the data consistency setting.Byte access by one MPM data transfer must be carried out in direct succession, as the MPM control logic cancels the MPM access inhibition feature after a certain time (timeout). The driver software functions automatically allow for this; so you do not need to take any special precautions

in your application program.

2.4.3 Interrupt Functions

Access methods with different access protocols are defined between the MPM nodes. The MPM hardware supports the protocols with *handshake interrupts*. In addition, each node interface has an interrupt line indicating the system failure of one of the nodes. Both daughterboards can interrupt the host PC by means of the *host interrupt*.

2.4.4 Voltage Monitoring, Reset System

The reliable operation of the IBS controller board requires a sufficient operating voltage supply (5V) via the AT bus. When the supply falls short of the minimum voltage, an integrated voltage monitoring feature triggers a reset of the IBS controller board. This prevents malfunctions on the InterBus-S due to undefined conditions while the controller board is switched on/off, and due to voltage dips. A reset of the controller board is triggered when the following conditions are met:

- The voltage monitoring circuitry trips
- Reset signal on the AT bus (can be disabled for IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T, see Figure 2-1, item 15)
- Operation of the reset button on the front plate (PC card holder) or of the reset button at the upper edge of the controller board

To prevent an unintentional operation of the reset button on the front plate, it is located in a concealed position (see Figure 2-2). Both buttons can also be disabled by removing a jumper (see Figure 2-1).

Table 2-5: Function of the jumper for the reset button

Jumper S105	Reset button
Installed	enabled
Not installed	disabled

This reset affects the following functional units:
- the AT bus interface of the controller board
- the MPM control logic
- the two daughterboards

The daughterboards can also be reset separately by the software.

2.4.5 Watchdog for Monitoring the Host PC

While the daughterboards have their own watchdog circuits for monitoring their operation, a watchdog for monitoring the host PC was incorporated into the AT bus interface on the IBS controller motherboard. The operation of the watchdog is described in detail in Section 4.

2.4.6 Power Supply

The controller board is supplied with power from the 5V system voltage of the PC, via the edge connectors. With certain flash EPROM types on the coprocessor board, the 12V supply of the AT bus is required for programming the flash EPROMs (current consumption: approx. 40mA).

2.4.6.1 Electrically Isolated IBS Power Supply

The driver components require an isolated power supply for the electrically isolated operation of InterBus-S. The motherboard is equipped with a DC/DC converter, which generates this voltage from the 5V system voltage of the PC. It provides an output voltage of 5 V at a maximum current of 200mA, which is capacitively coupled to PE for reasons of noise elimination.

Ground the controller board via the ground pin (Figure 2-2).

Section 3

Technical Description of the Coprocessor Boards

Only for IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T.

This section provides information on

- the structure and the components of the coprocessor board.

3 Technical Description of the Coprocessor Boards

(for IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T)

3.1 Short Description

The coprocessor boards (abbreviated: COP) are industrial PCs optimized for control tasks, in the form of daughterboards for the controller motherboards IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T.

Table 3-1: Overview of coprocessor boards

	IBS PC CB/COP/I-T	IBS PC CB/RTX486/I-T
Coprocessor board	COP386	COP486
CPU	386SX-25, MHz	486 SXLC-40, 40 MHz double clock, 8 Kbytes of cache
Main memory	2 Mbytes	2 Mbytes
Static RAM (battery backed-up)	128 Kbytes	128 Kbytes - as RAM disk (D:\)
EPROM 1	128 Kbytes of EPROM for operating system	256 Kbytes of flash EPROM: - 128 Kbytes for operating system - 128 Kbytes for device driver and application
EPROM 2	-	256 Kbytes of flash EPROM for application
Operating system	TDOS	RTXDOS
Access from the PC via	Development environment TDOS-PRO*	Terminal program DPCON.EXE (is on the tool diskette)

* The development environment TDOS-PRO (IBS PC COP SWT) is not compatible with the COP486 of the IBS PC CB/RTX486/I-T, nor is it required for it.

This coprocessor board is used to relieve the host PC processor of some of its tasks; this contributes to considerable performance gains of the PC system.

The central functional unit of the COP is an Intel-compatible microprocessor. The coprocessor board is accessed either via the Multi-Port Memory (MPM) or via a PC-compatible serial interface. The coprocessor boards are essentially PC-compatible (with the exception of the keyboard, the monitor and the hard disks), which allows programming with standard tools such as Turbo Pascal or Turbo C. The 2 Mbyte main memory provides enough space even for extensive software projects. The coprocessor board ensures the unrestricted use of the EMS features familiar from PCs.

128 Kbytes of battery-backed-up static RAM (SRAM), and, on the COP486, ad-

ditional 384 Kbytes of flash EPROM (128 Kbytes in flash EPROM 1 and 256 Kbytes in flash EPROM 2) are available for long-term storage of information or programs. The flash EPROM contains also the operating system (T-DOS) of the coprocessor board.

The coprocessor board operating system also is in flash EPROM 1.

The coprocessor board incorporates a quartz-controlled real-time clock (PC-compatible) for time-dependent control of machines via InterBus-S.

To ensure a high degree of data integrity in industrial applications, the coprocessor boards feature a sophisticated watchdog and reset circuitry. It enables the user to detect system errors at an early stage and to shut down equipment if necessary. After the protective mechanism has been enabled, the coprocessor boards are able to restart the application program by themselves.

The interrupt control feature on the coprocessor board informs other MPM nodes (host PC, IBS master) about any functional errors. Two LEDs on the front plate of the motherboard provide visual information on the status of the coprocessor board.

The *TDOS* and *RTXDOS* operating systems (DOS-compatible) provide a software basis which is specially adapted to the requirements of the coprocessor boards. This makes it possible to implement existing DOS applications on the coprocessor board. For real-time applications, additional real-time multitasking operating systems may be used (on request).

In the software development stage, the data transmission between the operating system TDOS of the COP386 and the development tool *TDOS-PRO (IBS PC COP SWT,* Order No. 27 52 12 3*)* can be effected via the MPM or the serial interface of the COP.

Using the *RFSERVER.EXE* program, which is to be started directly on the host, the operating system *RTXDOS* of the COP486 can directly access the host mass storage units (floppy drives, hard disks). *RFSERVER.EXE* is on the diskette supplied with this manual.

Table 3-2: COP 486 drives

Drive	Data carrier	Use
A:\	128 Kbytes in flash EPROM1	Device drivers and application programs
B:\	—	Not used
C:\	256 Kbytes in flash EPROM2	Application programs
D:\	128 Kbytes of static RAM (battery-backed-up)	Application programs, data
E:\	Drive C:\ of the PC (host hard disk)	Application programs, data

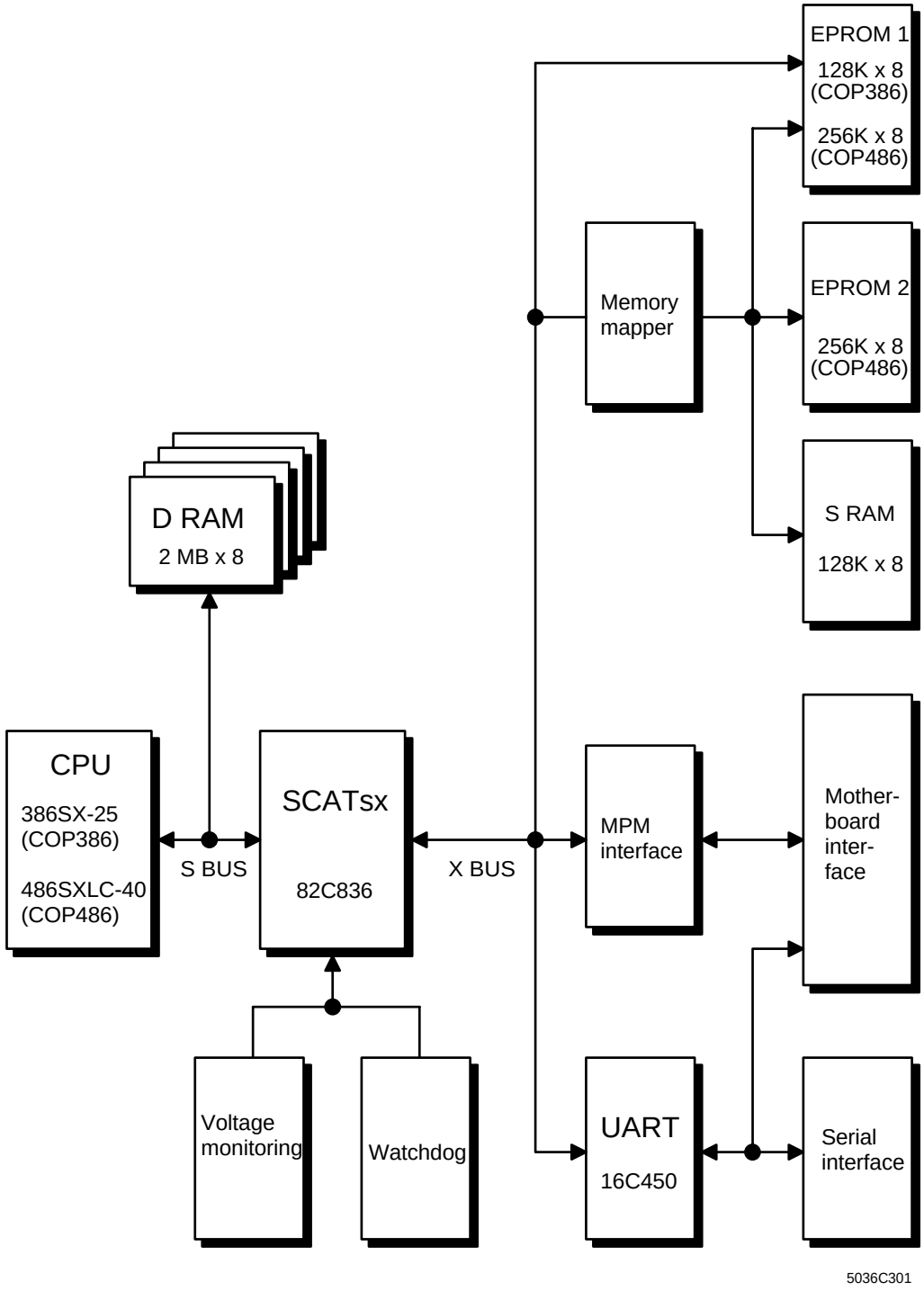


Figure 3-1: Coprocessor board block diagram

3.2 Mechanical Design

For the connection with the motherboard, the coprocessor board (COP) has a 58-position male connector on each side. The correct orientation for the installation of the COP results from the asymmetrically located drill holes (one on the coprocessor board and one on the motherboard). On the underside of the COP there is a four-position connector for the connection of a commercially available PC battery pack, and a 10-position connector for the serial interface.

To achieve a high flexibility of the COP, a feature for performing configurations via DIP switches or jumpers has not been implemented. The hardware-specific settings are automatically performed by the driver software.

3.3 Coprocessor Board Interfaces

This section describes the various coprocessor board interfaces and their signal assignment.

3.3.1 Motherboard Interface

Functions of the interface:

- Data exchange between the motherboard and the coprocessor board. The data width is 16 bits. All data and address lines are buffered.
- Supply of control signals: interrupt control signals, write and read control, reset and watchdog control.
- Supply and special lines: voltage supply, battery back-up, LED control, RS-232 interface.

3.3.2 Serial Interface

The coprocessor board has a serial interface (RS-232 level), which corresponds to the COM 1 of a standard PC, and which you can use, for example, for the following applications:

- If the development environment *TDOS-PRO (IBS PC COP SWT)* is not to access the COP386 by way of the MPM. For more detailed information refer to the manual for the development environment *TDOS-PRO*, which is supplied with the software.
- For remote debugging (e.g.: with Turbo/Borland C remote debugger). Please refer to the manual of your compiler for more detailed information.
- Own applications

Adapter cable *IBS PC COP RS 232 CAB*

This fully PC-compatible interface is implemented as a 10-position male connector on the edge of the coprocessor board. It can be connected to a subminiature D connector using the adapter cable *IBS PC COP RS232 CAB*. Mount the subminiature D connector on, for example, an additional PC board holder (free slot). The adapter cable is available under Order No. 27 51 65 8. It is also supplied with the development environment *TDOS-PRO (IBS PC COP SWT*).

The female connector of this adapter cable is identified by a polarizing bump next to pin 5.

- If the 10-position male connector is located in a polarized connector shell, install the female connector in the shell so that the polarizing bump faces the motherboard (see Figure 3-2).
- If you have an earlier coprocessor board without a polarized connector shell, mount the female connector on the shell so that the bump faces the copro-

cessor board (rotated by 180° compared with the drawing in Figure 3-2).

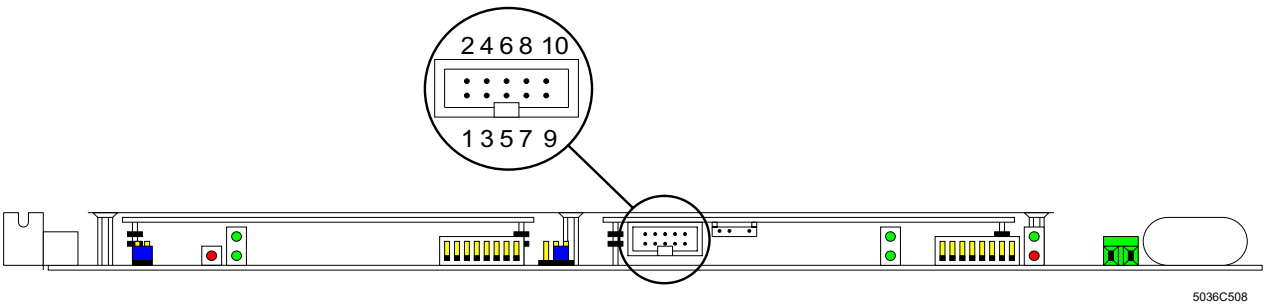


Figure 3-2: Pin arrangement in the COP connector shell

Table 3-3: Adapter cable assignment

Coprocessor board pin assignment	Function	Signal	Subminiature D9 pin assign-ment
1	Data Carrier Detect (input)	DCD	1
2	Data Set Ready (input)	DSR	6
3	Received Data (input)	RXD	2
4	Request To Send (output)	RTS	7
5	Transmitted Data (output)	TXD	3
6	Clear To Send (input)	CTS	8
7	Data Terminal Ready (output)	DTR	4
8	Ring Indicator (input)	RI	9
9	Signal Ground	GND	5
10	Unused	-	-

Development cable

The connection of the serial interface (subminiature D connector) of the COP386 and of the PC on which the development environment *IBS PC COP SWT (TDOS-PRO)* is running is effected with a development cable (zero modem cable) as shown below.

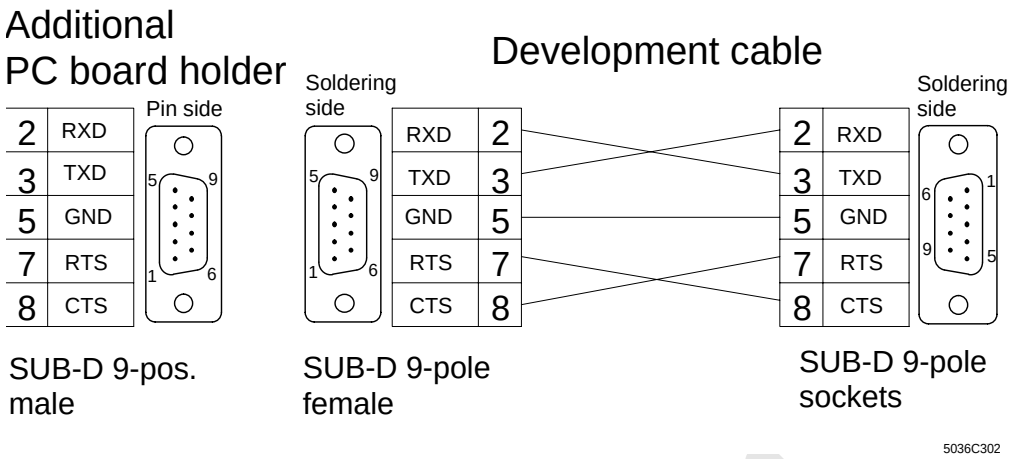


Figure 3-3: Development cable for the COP386

The development cable is supplied with the development environment TDOS-PRO (IBS PC CB COP SWT, Order No. 27 52 12 3).

3.4 Coprocessor Board Functional Units

The functionality of the coprocessor board is divided into seven main functional elements:

- Processor/chipset
- Memory (DRAM; SRAM; EPROM)
- MPM interface (Multi-Port Memory)
- Memory mapper
- Serial interface (UART)
- Voltage monitoring and reset system
- Power supply

The following sections describe these funtional units, their settings and meanings in detail.

3.4.1 Processor/Chipset

The processor core is a 386SX-25 for the COP386 and a 486SXLC40 microprocessor for the COP486, both in connection with the single-chipset 82C836.

3.4.1.1 Chipset Components

The 82C836 component is a highly integrated single-chip AT chipset. It contains all circuitry components required for a PC, such as:

- one 146818-compatible real-time clock with 114 bytes of CMOS-RAM
- two 8237-compatible DMA controllers
- two 8259-compatible interrupt controllers
- one 8254-compatible counter/timer
- one 82284-compatible clock generator and ready interface
- one 82288-compatible bus controller
- one DRAM controller / DRAM refresh controller
- four EMS page registers (LIM EMS 4.0 and 3.2)

3.4.1.2 Coprocessor Board I/O Address Area

Besides the usual standard addresses of the IBM PC, the I/O address area of the coprocessor board has some additional special functions. Various watchdog functions are enabled in the address area from 200_{hex} to $2BF_{hex}$ (according to IBM reserved for the prototype board). The address area from 280_{hex} to $2BF_{hex}$ contains the mapping register (a register for controlling controller-specific memory functions) including some bits for special functions such as for watchdog control.

Table 3-4: I/O address area of the coprocessor board

I/O address:			Function:
$03C0_{hex}$	to	$03FF_{hex}$	Area for the COM1 serial interface
0340_{hex}	to	$03BF_{hex}$	Unused
0300_{hex}	to	$033F_{hex}$	Watchdog Clear
$02C0_{hex}$	to	$02FF_{hex}$	Unused
0280_{hex}	to	$2BFF_{hex}$	Mapping register
0240_{hex}	to	$027F_{hex}$	Watchdog enable
0200_{hex}	to	$023F_{hex}$	Watchdog trigger
$00E0_{hex}$	to	$01FF_{hex}$	Unused
$00C0_{hex}$	to	$00DF_{hex}$	DMA controller 2
$00A0_{hex}$	to	$00BF_{hex}$	Interrupt controller 2 (Slave)
0080_{hex}	to	$009F_{hex}$	DMA page register / NMI mask
0070_{hex}	to	$007F_{hex}$	Real-time clock
0060_{hex}	to	$006F_{hex}$	System register
0040_{hex}	to	$005F_{hex}$	Timer / counter
0020_{hex}	to	$003F_{hex}$	Interrupt controller 1 (master)
0000_{hex}	to	$001F_{hex}$	DMA controller 1

The base address for the serial interface COM2 is, as usual for IBM-compatible PCs, at the address $03F8_{hex}$.

3.4.1.3 Coprocessor Board Interrupt Assignment

The interrupt system is also mainly identical to the usual standard of an IBM PC. For the communication with the MPM a so-called handshake interrupt is used, which has the interrupt number 5. In the event of a system error of another MPM node (host PC, IBS master), a *non-maskable interrupt* (NMI) is triggered on the coprocessor board. When, however, an error occurs on the coprocessor board, an interrupt signal to the other nodes is generated if the watchdog is enabled. This signal is fed to the mother board interface and is referred to as *SRQ2L*. The following table outlines the interrupt assignment of the coprocessor board.

Table 3-5: Interrupt assignment of the coprocessor board

Interrupt	Function
IRQ0	Timer channel 0
IRQ1	Unused
IRQ2	Slave interrupt controller
IRQ3	Unused
IRQ4	COM1
IRQ5	Handshake interrupt MPM
IRQ6	Timer channel 2
IRQ7	Unused
IRQ8	Real-time clock
IRQ9	Unused
IRQ10	Unused
IRQ11	Unused
IRQ12	Unused
IRQ13	Unused
IRQ14	Unused
IRQ15	Unused

3.4.2 Coprocessor Board Memory

The memory segmentation is as follows:

- 128 Kbytes of static RAM (SRAM), battery-backed-up with a data width of 8 bits
- 2 Mbytes of DRAM with a data width of 16 bits

3.4.2.1 EPROM

Two 32-position PLCC EPROM sockets for 128 Kbyte or 256 Kbyte EPROMs or flash EPROMs are provided on the coprocessor board.

IBS PC CB/COP/I-T

On delivery, socket 1 is equipped with a 128 Kbyte EPROM for the operating system. Socket 2 is not fitted with an EPROM.

IBS PC CB/RTX486/I-T

On delivery, socket 1 and socket 1 are equipped with 256 Kbytes of flash EPROM each.

3.4.2.2 Static RAM

128 Kbytes of static RAM (SRAM) are available on the coprocessor board for storing data (e.g. equipment states). This memory is battery-backed and has a width of 8 bits.

IBS PC CB/COP/I-T

The SRAM on the COP386 can be addressed via the memory segment $B0000_{hex}$ in two 64 Kbyte blocks. The access is via driver software functions.

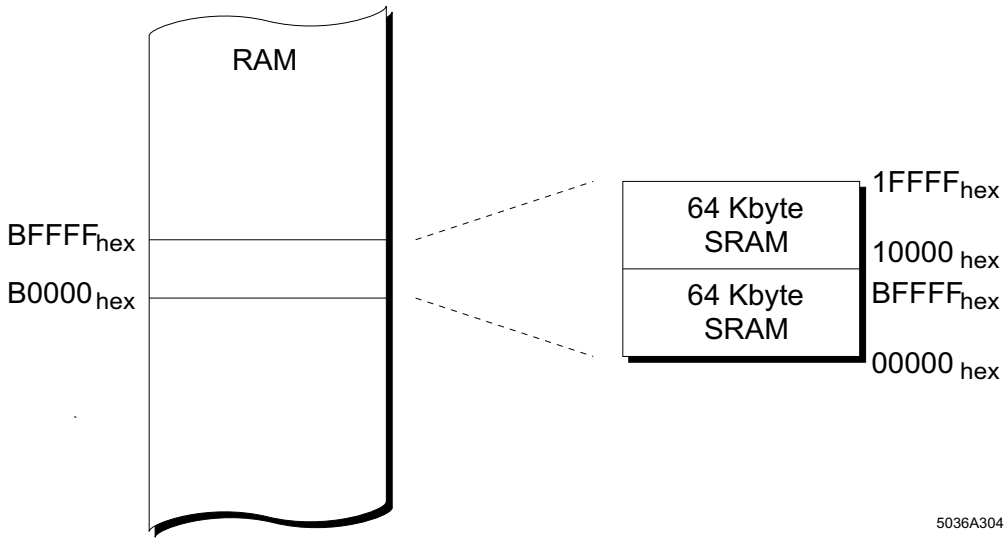


Figure 3-4: SRAM segmentation on the COP386 of the IBS PC CB/COP/I-T

IBS PC CB/RTX486/I-T

The SRAM on the COP486 is accessed as drive D:\ and, therefore, can be used as RAM disk by means of usual DOS functions.

The program *RDRCONF.EXE* is used to parameterize the type of access to the SRAM of the COP486. You can, for example, also set the addressing described for the COP386 in two 64 Kbyte blocks. This manual comes complete with *RDRCONF.EXE*.

3.4.2.3 Dynamic RAM

The coprocessor board contains 2 Mbytes of dynamic RAM (DRAM) as main memory. The main memory is operated with 0 RAM wait states and in the page interleave mode, which increases the processing speed.

As the coprocessor board works in the real mode because there is no keyboard controller, the use of the second megabyte (100000_{hex} to $1FFFFF_{hex}$) is only possible when the EMS functions are used. For this purpose there is an EMS driver on the tool diskette.

Table 3-6: DRAM segmentation

Megabyte	Adress area	Bank	Allocation	Supplement
2.	100000_{hex} to $1FFFFF_{hex}$	10 to 1F	DRAM	XMS/EMS memory
1.	$0F0000_{hex}$ to $0FFFFF_{hex}$	F		Bios
	$0E0000_{hex}$ to $0EFFFF_{hex}$	E		
	$0D0000_{hex}$ to $0DFFFF_{hex}$	D		reserved
	$0C0000_{hex}$ to $0CFFFF_{hex}$	C		
	$0B0000_{hex}$ to $0BFFFF_{hex}$	B	Mapping window	
	$0A0000_{hex}$ to $0AFFFF_{hex}$	A	DRAM	Main memory/ download RAM
	090000_{hex} to $09FFFF_{hex}$	9		
	080000_{hex} to $08FFFF_{hex}$	8		
	070000_{hex} to $07FFFF_{hex}$	7		
	060000_{hex} to $06FFFF_{hex}$	6		
	050000_{hex} to $05FFFF_{hex}$	5		
	040000_{hex} to $04FFFF_{hex}$	4		
	030000_{hex} to $03FFFF_{hex}$	3		
	020000_{hex} to $02FFFF_{hex}$	2		
	010000_{hex} to $01FFFF_{hex}$	1		Main memory
	000000_{hex} to $00FFFF_{hex}$	0		TDOS system area

The 4 Kbyte memory area from $B0000_{hex}$ to $BFFFF_{hex}$ is configured as a mapping window, via which the driver software functions access the MPM, the static RAM and the EEPROMs.

Do not directly access the coprocessor board DRAM in the area of the mapping window!

The driver software functions select via the mapping register in the I/O area of the coprocessor board which memory is mapped via the mapping window.

3.4.3 MPM Interface

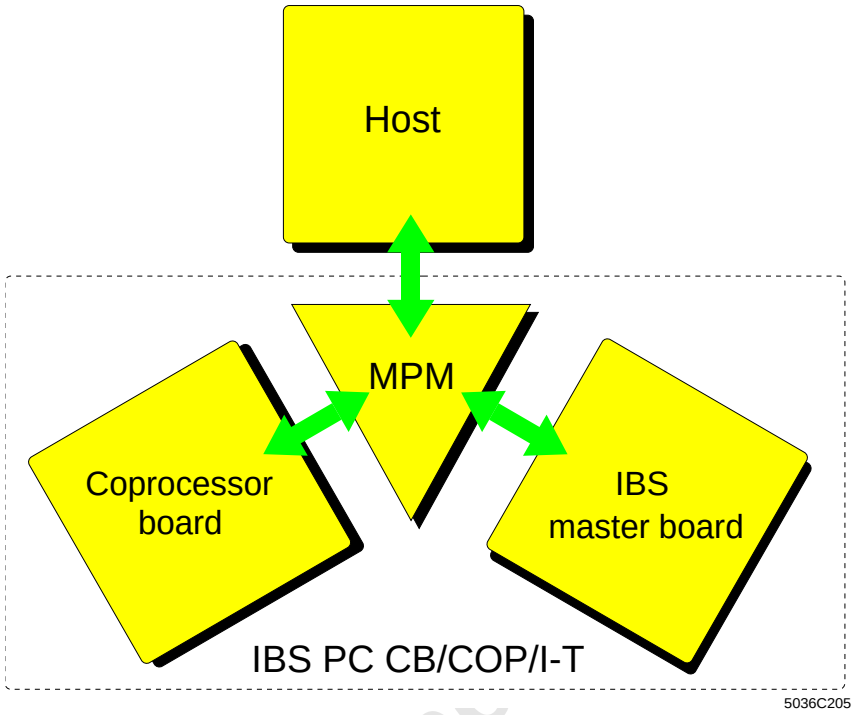


Figure 3-5: The MPM as the central interface

The MPM is the central interface between the host PC, IBS master (MA) and coprocessor board (COP). To ensure a problem-free communication between the MPM nodes, all data and address lines of the COP are buffered. The buffers are only enabled when the coprocessor board is granted the right of access. The right of access is granted by a priority control feature in the MPM logic (on the motherboard). After the MPM request by the coprocessor board and before the motherboard grants the right of access, the COP processor is in the *Halt state*.

When the coprocessor is granted the right of access, the data and address buffers are enabled, and the data transfer begins with a data consistency of 8, 16, 32 or 48 bits, according to the setting of the MPM address (the setting is done by means of the mapping register). When the data consistency is 16, 32 or 48 bits, multiple 8-bit accesses are carried out.

Please note the following when accessing the MPM. The IBS master board places its data in the MPM in the Motorola format (68xxx family), whereas the host and COP processors expect data in the Intel format. These two formats have opposite orders of byte addresses (numbering). For MPM accesses use supplied macros for data conversion, which exchange the low byte and the high byte (see the Driver Software Manual *IBS PC CB SWD UM E*, which is supplied with the driver software *IBS PC CB SWD*).

3.4.4 Mapping Register

The memory mapper plays a central part in the coprocessor hardware design. It is located in the I/O area of the coprocessor board. It is an 8-bit register, which enables the switching over of the individual memory areas to the segment $B0000_{hex}$-$BFFFF_{hex}$. This memory area provides access to the EPROMs, the RAM and the MPM window. The access is performed by means of driver software functions.

3.4.5 Coprocessor Board Serial Interface

The serial interface of the coprocessor board allows the software development and the program test without the use of the MPM interface. Thus you can, for example, carry out remote debugging for quick error detection of your own software applications using the Borland Turbo Debugger. The serial interface is configured as COM 1, which is fully PC-compatible. The base address is, as usual for IBM-compatible PCs, $3F8_{hex}$, and the interrupt IRQ 4 is used. These settings cannot be configured.

3.4.6 Coprocessor Board Security Units

The voltage monitoring and watchdog units ensure in connection with the reset system a defined behavior of the coprocessor board. In addition, they protect the system in the event of a supply voltage dip or in the event of application program runtime errors from an uncontrolled response.

3.4.6.1 Coprocessor Board Voltage Monitoring

The coprocessor board voltage monitoring feature monitors +5 V supply. If the supply voltage falls below 4.65 V, the voltage monitoring feature causes a system reset, and the Chip Enable line of the CMOS-RAM is disabled. From this moment onwards, the external battery voltage is used for buffering the real-time clock and the static RAM. The error-free operation of the voltage monitoring circuit is ensured for a voltage range from 1.0 V to 5.25 V.

3.4.6.2 Coprocessor Board Reset System

The coprocessor board has an extensive reset system for the internal hardware. A reset may be automatically initiated by the internal functional units of the voltage monitoring feature and the watchdog system, or a reset can be carried out by an MPM signal. The chipset reset logic then spefically resets the individual coprocessor board components such as the processor, the serial interface and the mapping register.

3.4.6.3 Coprocessor Board Watchdog

The watchdog is used to monitor whether the application program operates problem-free. After the watchdog has been enabled, it must be triggered at regular invervals. If this trigger pulse fails to be issued for 125 ms, the coprocessor board carries out a restart. After successful booting of the system,

the user can determine the watchdog status by means of a function. The tripping of the coprocessor watchdog is indicated to other devices of the IBS system via an interrupt line. The watchdog status is reset when the host PC is switched off or when the respective function is called. The start of the watchdog and the triggering are also carried out by functions. The trigger time is invariably 125 ms. After a system reset the monitoring circuit is always disabled. Once it has been enabled, a watchdog cannot be disabled by the software.

Watchdog control functions:

- EnableWatchDog()	Watchdog enabling
- TriggerWatchDog()	Watchdog triggering
- GetWatchDogState()	Watchdog status bit inquiry
- ClearWatchDog()	Watchdog status bit resest

The driver software manual *IBS PC CB SWD UM E* describes these functions in detail.

3.4.7 Coprocessor Board Real-time Clock

The coprocessor boards have a quartz-driven real-time clock. It is backed up by the battery pack on the motherboard.

The real-time clock is PC-compatible.

- The development environment *TDOS-PRO (IBS PC COP SWT)* provides the program *RTCINIT.EXE* to set the date and time on the COP386. Download the program with the development environment to the coprocessor board and start it. Then you can set the date and the time.
- On the COP486, RTXDOS allows the setting of date and time with the usual DOS commmands.

Use of timers

Like on a normal PC, timer 0, which normally calls interrupt 1C 18 times in a second, can also be set to different values. The timer is, as usual, accesssed via the I/O addresses 40_{hex} to 43_{hex}. You will find numerous detailed articles on the programming of timers in the PC reference literature.

3.4.8 Coprocessor Board Power Supply

The coprocessor board was designed so that a +5 V supply is normally sufficient. The situation is different when flash EPROMs with a programming voltage 12 V are used. With this type of EPROM, a 12 V supply is required in addition to the 5 V. This voltage is provided on the mother board of the IBS controller. Thanks to the CMOS components used, the total current consumption of the coprocessor board is only 400 mA. With the flash EPROM programming with a programming voltage of 12 V, an additional current consumption of approx. 40 mA must be taken into account.

3.4.8.1 Battery Back-up

For the back-up of the RAM memory on the coprocessor board, the RAM memory on the coprocessor board has a standard battery pack as used in PCs for BIOS setting back-up. The battery pack provides a voltage of 6V and has a life of approx. 2.5 years. When replacing the battery pack, please use one of the same size.

Section 4

Installation and First Startup

This section provides information on

- address settings on the controller board;
- the installation of the controller board in your PC;
- the installation of the driver software.

4 Installation and First Startup

The following subsections describe in turn all preparations required for controller board startup.

The controller boards can be used in 100% IBM-compatible PCs (AT, 80386, 80486 etc.). The required main memory size depends exclusively on the size of the application program.

The DIP switch settings are only read in when the controller board boots. After the setting has been changed, reset the controller board to make the change effective.

The controller boards may possibly be equipped with different types of DIP switches. Therefore, please note the marking for ON/OFF, which is printed directly on the DIP switches.

The switches that are not relevant for the described function are shown in gray.

4.1 Address Setting

4.1.1 Base Address in the I/O Area of the PC (I/O Address)

The controller boards are set with 8 bytes in the I/O area of the PC. The base address is the address of the first of these 8 bytes. It is set on the DIP switches shown in Figure 4-1. The default setting is 120_{hex} and normally does not need to be changed. Please ensure that the address area set is not already used by other components of your PC.

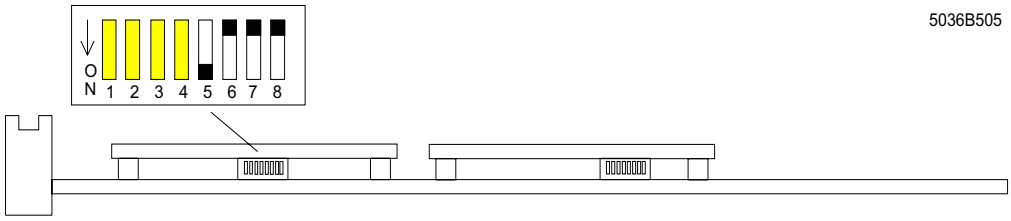


Figure 4-1: DIP switches for setting the I/O address

16 addresses in the I/O address area of the PC are available for the PC I/O address; from these addresses you can select the desired one by means of DIP switches. The following Table 4-1 describes the permissible PC I/O addresses with the required settings.

Switches 3 and 4 are *OFF* on delivery. Please do not change these settings!

Table 4-1: Possible settings for the base address in the I/O area

PC IO address	Switch 5	Switch 6	Switch 7	Switch 8
100_{hex}	OFF	OFF	OFF	OFF
120_{hex} (default)	**ON**	OFF	OFF	OFF
140_{hex}	OFF	**ON**	OFF	OFF
160_{hex}	**ON**	**ON**	OFF	OFF
180_{hex}	OFF	OFF	**ON**	OFF
$1A0_{hex}$	**ON**	OFF	**ON**	OFF
200_{hex}	OFF	**ON**	**ON**	OFF
220_{hex}	**ON**	**ON**	**ON**	OFF
240_{hex}	OFF	OFF	OFF	**ON**
280_{hex}	**ON**	OFF	OFF	**ON**
$2A0_{hex}$	OFF	**ON**	OFF	**ON**
300_{hex}	**ON**	**ON**	OFF	**ON**
320_{hex}	OFF	OFF	**ON**	**ON**
340_{hex}	**ON**	OFF	**ON**	**ON**
380_{hex}	OFF	**ON**	**ON**	**ON**
$3A0_{hex}$	**ON**	**ON**	**ON**	**ON**

Up to four controller boards can be used in one PC. Set the same base address on all four controller boards. The controller boards are distinguished by means of the board number.

4.1.2 Board Number (Board No.)

If several controller boards are operated in one host (PC), the boards are distinguished by means of the board number (1 to 4) set with DIP switches on the motherboard. In this case, the same base address must be set on the controller boards. The board number then effects an automatic offset of 0_{hex}, 8_{hex}, 10_{hex} or 18_{hex} relative to the base address in the PC I/O area.

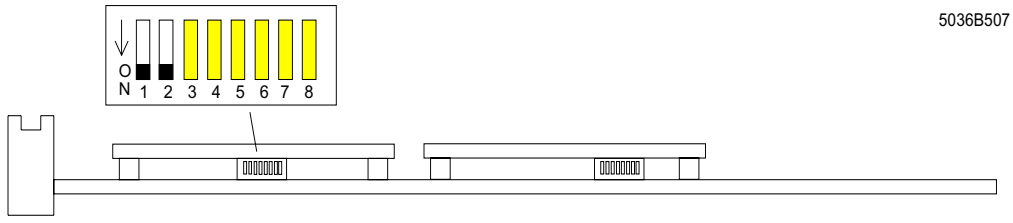


Figure 4-2: Switches for setting the board number (board no.)

If you want to use four controller boards, 32 bytes (4 times 8 bytes) in the PC I/O area from the set base address onwards must be free.

Table 4-2 shows the board number, the resulting (automatic) offset relative to the common base address, and the related DIP switch setting.

Table 4-2: Setting of the board number of controller boards

Switch 1	Switch 2	Board number	Corresponding to offset relative to base address
ON	ON	1	00_{hex}
OFF	ON	2	08_{hex}
ON	OFF	3	10_{hex}
OFF	OFF	4	18_{hex}

If you use only one controller board, ensure that board number *1* has been set (default).

If, for example, you set a base address of $3A0_{hex}$ in the I/O area, using DIP switches 5 to 8, the resulting address area for the controller board with the board number 4 ranges from $3B8_{hex}$ to $3C0_{hex}$:

Base address **+** offset **+** I/O area for controller board 4

$3A0_{hex}$ **+** 18_{hex} **+** 8_{hex} **=** $3C0_{hex}$

Ensure that the address area required by the controller boards is not used by other components of your PC.

4.2 Setting the Boot Configuration

Using DIP switches, you can select particular boot configurations on the motherboard. The DIP switch settings are only read in when the controller board is booted. After the setting has been changed, always carry out a controller board reset to make the change become effective.

4.2.1 IBS Control

(Only IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T)

When the IBS controller boards with coprocessor board are used, the IBS master board can be controlled either by the host (PC) or by the coprocessor board (COP). Only from here can commands (e.g. bus start) be issued and the IBS output data be changed. Using DIP switch 1, determine whether the IBS master board is to be controlled by the host or by the coprocessor board. The IBS input data can in any case be read by the host as well as by the COP.

When using the IBS PC CB/I-T (without coprocessor board) always set switch 1 to *ON*.

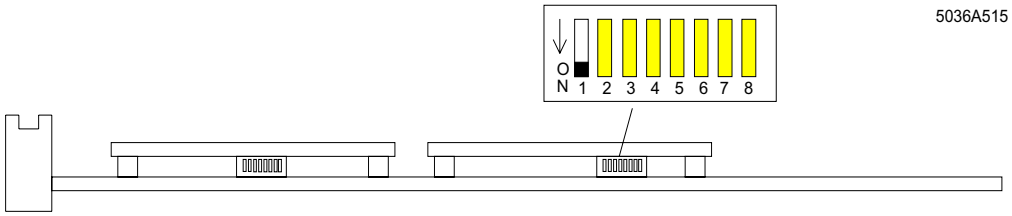


Figure 4-3: IBS control switches

Table 4-3: Control of the IBS master board by the host or by the COP

Switch 1	IBS master board control by the
ON (default)	Host (PC)
OFF	Coprocessor board

The controller board reads out switch 1 only after a hardware reset. After changing the switch setting always carry out a hardware reset to make the change become effective.

4.2.2 InterBus-S Startup Behavior (IBS Autostart)

The DIP switch setting defines the InterBus-S startup behavior after IBS master board booting. If the switch is OFF, the IBS data cycles are automatically started if an operable bus configuration has been connected. From this moment onwards the controller board transfers data from the MPM to the I/O periphery and vice versa.

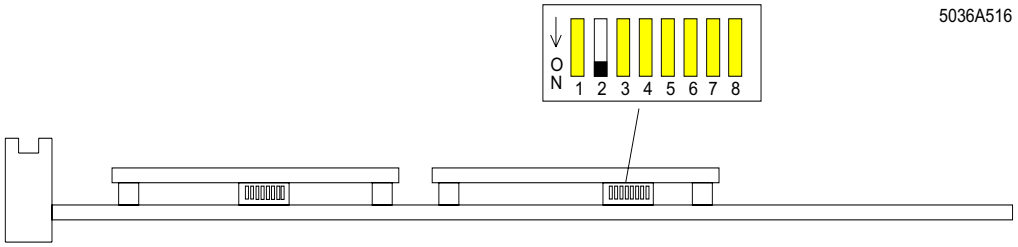


Figure 4-4: Switch for the startup behavior (IBS Autostart)

Table 4-4: Definition of the startup behavior

Switch 2	Startup behavior
ON (default)	No automatic start
OFF	Automatic start

If an error occurs on InterBus-S after the automatic start of the IBS data cycles, and this error causes InterBus-S to stop, the controller board waits for a manual hardware reset with the front plate reset button. In this case, the IBS data cycles cannot be restarted by the application program. Please consider this when using the automatic start of the IBS data cycles.

4.2.3 Automatic Program Start from EPROM (EPROM Start)

(only IBS PC CB/COP/I-T with TDOS)

Switch 6 is used to define under TDOS whether programs stored in the EPROM are to be automatically started after COP booting, or not. The automatic start feature should be disabled when, for example, the programs stored in the EPROM are faulty, causing the coprocessor board to crash immediately after booting.

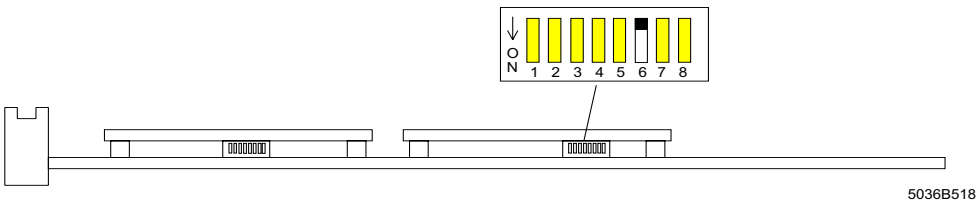


Figure 4-5: Switch setting for the program start from the EPROM (EPROM start)

Table 4-5: Program start from the flash EPROM

Switch 6	Programs from the EPROM
OFF (default)	Are not automatically started
ON	Are automatically started

Switches 3, 4, 5 and 7 of the IBS PC CB/COP/I-T are *OFF* on delivery. Do not change these settings!

4.2.4 Setting the Boot Disk

(Only IBS PC CB/RTX486/I-T with RTXDOS)

Switch 6 is used to determine the boot drive for the coprocessor board under RTXDOS (the drives of the coprocessor board are described in Section 2 and in the RTXDOS documentation).

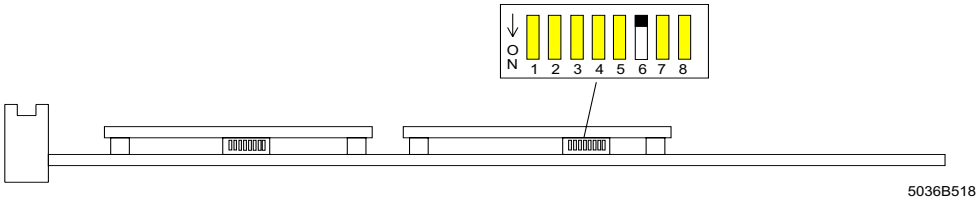


Figure 4-6: Switch for boot disk selection

Table 4-6: Selection of the boot drive for the coprocessor board

Switch 6	Boot drive
OFF (default)	A:\ (flash EPROM1)
ON	D:\ (static RAM)

Automatic start of programs under RTXDOS

Upon delivery of the IBS PC CB/RTX486/I-T, the root directory of drive A:\ (flash EPROM) contains an *AUTOEXEC.BAT* and a *CONFIG.SYS* with the basic entries. To start programs automatically after coprocessor board booting, proceed as follows:

1. Start the program *DPCON.EXE* (is on the tool diskette).
2. Copy the files *AUTOEXEC.BAT* and *CONFIG.SYS* from drive A:\ (flash EPROM) to drive D:\ (SRAM). The drives of the coprocessor board are described in Section 2 and in the RTXDOS documentation.
3. Using an editor, enter the programs that are to be started automatically into *AUTOEXEC.BAT* on drive D:\.
4. Using switch 6, set drive D:\ as boot drive.
5. Boot the coprocessor board. The programs entered in *AUTOEXEC.BAT* on drive D:\ are, if error-free, automatically started after the coprocessor board has booted.

Switching to a different boot drive is also useful when the programs called by the AUTOEXEC.BAT to be executed by the coprocessor board are faulty and, therefore, the coprocessor board crashes immediately after booting. By switching to the other boot drive, the coprocessor board can now boot without calling the faulty programs.

Programs that are to be started automatically should never be entered at the same time in *AUTOEXEC.BAT* on drive A:\ and on drive D:\ , as in that case you would not be able to prevent the call of faulty programs with switch 6.

Programs that are to be started automatically can be stored at the following locations:

- in flash EEPROM 1, which is to be selected as drive A:\ from the COP;
- in flash EEPROM 2, which is to be selected as drive C:\ from the COP;
- in the battery-backed-up SRAM, which is to be selected as drive D from the COP:\;
- on the host hard disk (C:\ ), which is to be selected as drive E:\ from the COP.

Switches 3 and 4 of the IBS PC CB/RTX486/I-T are *OFF* as default. Do not change these settings!

4.2.5 *RFSERVER* Boot Behavior (Wait for RFSERVER)

(Only IBS PC CB/RTX486/I-T mit RTXDOS)

The TSR program *RFSERVER.EXE* allows the coprocessor board to access the drives of the host PC. After the start of *RFSERVER* on the host, the coprocessor board can, for example, load programs from the host hard disk and read and store data from and to the disk. The connection between *RFSERVER* and *RTXDOS* is effected via the MPM.
The *RFSERVER.EXE* program is on the tool diskette.

While the coprocessor board is booting, *RTXDOS* attempts to establish a connection with the program *RFSERVER.EXE* on the host. As the coprocessor boards boots very fast, it is likely that *RFSERVER.EXE* on the host has not been started when a restart of the complete system is performed. Using switch 5 you can now determine whether in such a case the coprocessor board is to continue the boot procedure, or whether it is to wait until *RFSERVER.EXE* has been started on the host. For example, it is not recommended to wait for the start of *RFSERVER.EXE* in the stand-alone mode of the *IBS PC CB/RTX486/I-T*.

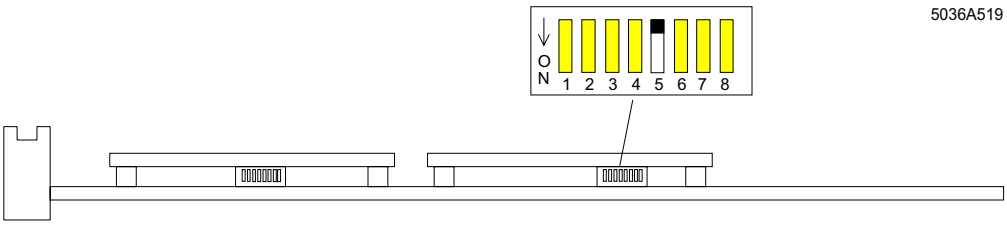


Figure 4-7: Switch for setting the *RFSERVER* boot behavior

Table 4-7: *RFSERVER* boot behavior of the coprocessor board

Switch 5	Boot behavior
OFF (default)	Coprocessor board waits for the start of *RFSERVER* when booting
ON	Coprocessor board does not wait for the start of *RFSERVER* when booting

4.2.6 *DPCON* Boot Behavior (Wait for DPCON)

(Only IBS PC CB/RTX486/I-T with RTXDOS)

The program *DPCON.EXE* makes it possible to use the host PC keyboard and monitor as a "terminal" for the coprocessor board. After the start of *DPCON.EXE* on the host, a DOS shell is available to the user on the host for the coprocessor board. The connection between *DPCON.EXE* and *RTXDOS* is effected via the MPM. The *DPCON.EXE* program is on the tool diskette.

While the coprocessor board is booting, it writes the screen outputs that are usual for a PC to the MPM. Using the program *DPCON.EXE* these outputs can be fetched and displayed on the host monitor.

If *DPCON.EXE* does not fetch the screen outputs from the MPM, the MPM area reserved for this purpose may become full. Using switch 7 you can now determine whether the coprocessor board continues the boot procedure in such a case, or whether it waits until the RAM area is free again.

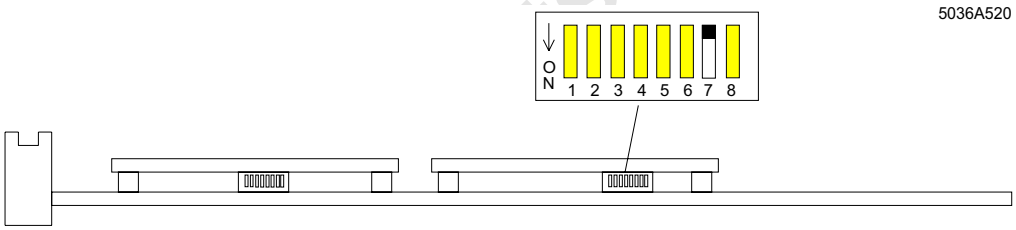


Figure 4-8: Switch for setting the DPCON boot behavior

Table 4-8: DPCON boot behavior

Switch 7	Boot behavior
OFF (default)	When the MPM area is "full", the coprocessor board continues with the boot procedure, although DPCON has not yet read the characters from the MPM. This may result in the loss of characters.
ON	When the MPM area is "full", the coprocessor board waits until DPCON has read the characters from the MPM before it continues with the boot procedure.

4.2.7 Data Transmission Between COP and Development Environment

(Terminal mode, only IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T)

The data transmission path (terminal mode) between the coprocessor board and

- the development environment *TDOS-PRO (IBS PC COP SWT)* or
- the terminal program *DPCON* (is on the tool diskette)

may be via the MPM or via the serial interfaces of the coprocessor board and of the PC. This is set with DIP switch 8.

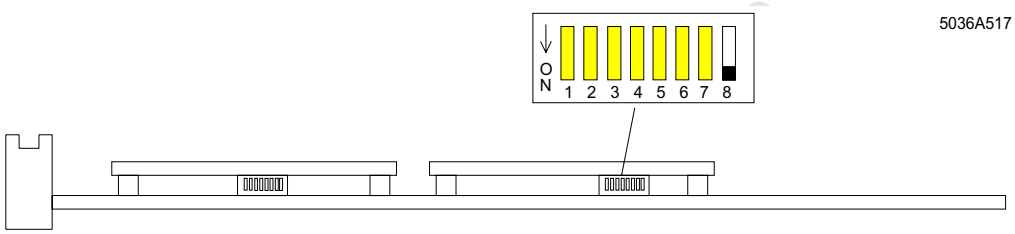


Figure 4-9: Switch for setting the data transmission path (terminal mode)

Table 4-9: Setting the terminal mode

Switch 8	Data transmission path
ON (default)	Via the MPM
OFF	Via the serial interfaces (RS232)

The development environment *TDOS-PRO (IBS PC COP SWT)* is not compatible with the COP486 of the *IBS PC CB/RTX486/I-T* and is not required for it.

If the connection is to be effected via the serial interfaces, these must be interconnected by means of the adapter cable *IBS PC COP RS 232 CAB* and the development cable (see Section 3). Both cables are supplied with the development environment *TDOS-PRO*.

Set switch 8 to ON if you want to use the serial interface of your coprocessor board for functions of your application program.

4.3 Jumper Settings

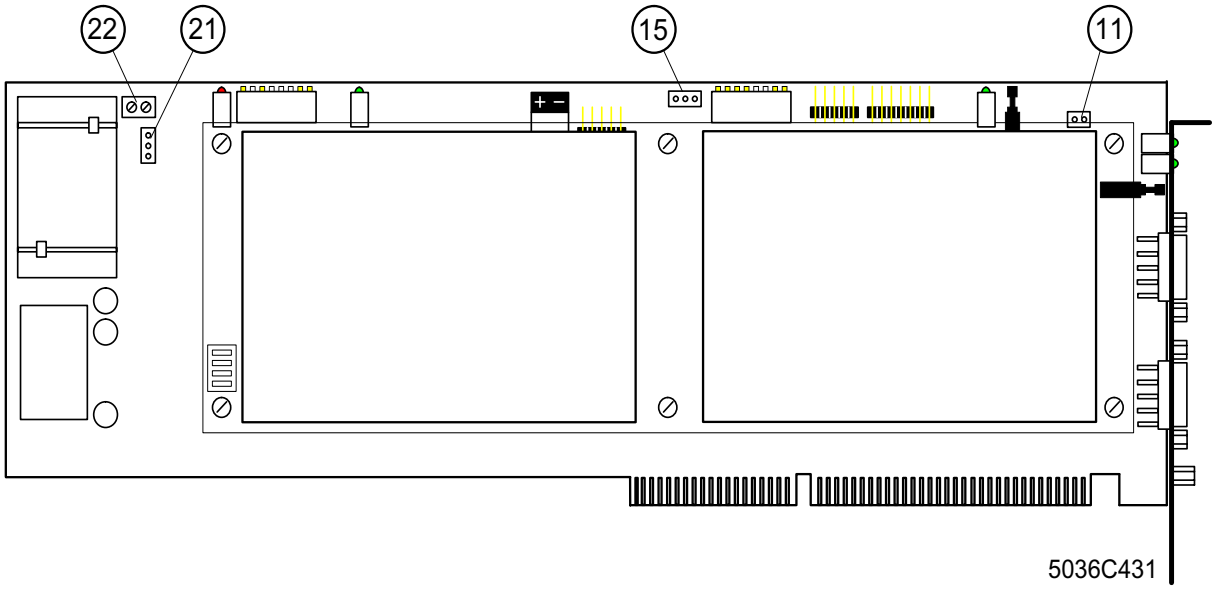


Figure 4-10: Jumpers for controller board parameterization

22 Terminal for feeding in the external supply voltage (not on IBS PC CB I-T)

The controller board is parameterized via the following jumpers:
21 Jumper for voltage supply selection (not on IBS PC CB I-T)
15 Jumper for separation from the PC hardware reset (not on IBS PC CB I-T)
11 Jumper for disabling the reset buttons

4.3.1 Power Supply Selection

(Only IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T)

The *Power Supply* jumpers are used to select whether the IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T controller boards are to be supplied with power via the host PC AT bus or via the terminals for external supply.

Table 4-10: Function of the power supply selection jumper (see labelling on the controller board)

***Power Supply* jumper**	**Supply voltage**
Installed in position "PC" (default)	From the host PC via the AT bus
Installed in position "Ext."	From an external power pack via the green CombiCon terminals next to the battery pack (see Figure 4-10, position 22)

4.3.2 Separation from the Host PC Hardware Reset (PC HW RESET)

(Only IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T)

Using the jumper *Enable/Disable PC HW RESET* (Figure 4-10, position 15) you can configure the controller boards IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T so that they are not affected by a hardware reset of the host PC.

Table 4-11: Function of the jumper for separation from the host hardware reset (see labelling on the controller board)

Jumper Enable/Disable PC HW RESET	Controller board reset behavior
Set to ON (enabled, default)	A host reset causes the controller board to boot.
Set to OFF (disabled)	A host reset does not affect the controller board.

4.3.3 Reset Button Disabling (Enable/Disable RESET Button)

To prevent any unintentional pressing of the reset buttons, the buttons can be disabled by removing the *Enable/Disable RESET Button* jumper (Figure 4-10, position 11).

Table 4-12: Function of the Enable/Disable RESET Button (see labelling on the controller board)

Jumper *Enable/Disable RESET Button*	Reset button
Enable (jumper installed, default)	Enabled
Disable (jumper removed)	Disabled

4.4 Connection of the Battery Pack

(Only IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T)

As default, the battery pack for supplying the SRAM and the real-time clock on the coprocessor board is not connected with the coprocessor board. Before installing the controller board in your PC, connect the connector of the battery pack to the male connector marked *SRAM Battery 6V* on the edge of the coprocessor board.

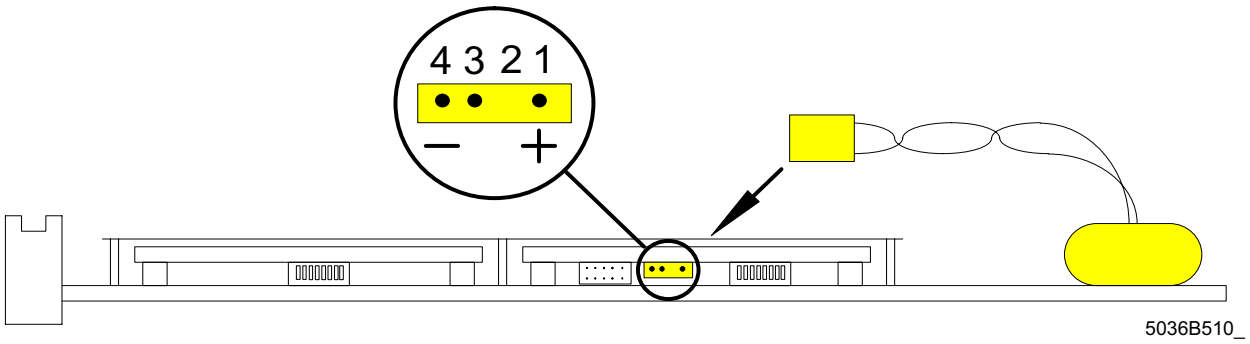


Figure 4-11: Position of the connector for the battery pack

Table 4-13: Pin assignment of the connector for the battery pack

Pin	Signal	Color of the connecting cable
1	+6V battery	Red
2	Code pin	-
3	GND	-
4	GND	Black

The battery pack has an operating life of approx. 2.5 years

4.5 Installation of the Controller Board in the PC

To avoid damage to the PC board and the host, the controller board may only be installed or removed when the PC is not supplied with power.

Ensure a sufficient clearance between the controller board and other PC boards. It is important to avoid any contact between the PC boards. Install controller boards only in AT bus slots with sufficient space (e.g. not above space-consuming memory components, SIMMs, etc.).

The controller boards require a deep AT bus slot in the host (PC).

The PC power supply must be sufficiently rated, especially for the operation of several controller boards in one PC.

Table 4-14: Controller board power consumption

Controller board	Supply
IBS PC CB/I-T	1.2 A (5V)
IBS PC CB/COP/I-T	1.5 A (5V)
IBS PC CB/RTX486/I-T	1.5 A (5V)

Ground the controller board via the ground pin on the PC board holder.

4.5.1 Serial Interface of the Coprocessor Board

(Only IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T)

The coprocessor board has a serial interface (RS 232 level), which corresponds to COM 1 of a standard PC. This fully PC-compatible interface is provided on the edge of the coprocessor board as a 10-position male connector and can be connected with the adapter cable *IBS PC COP RS 232 CAB* to a subminiature D connector, which you install, for example, on an additional PC board holder (free slot). The adapter cable can be ordered under Order No. 27 51 65 8 . It is also supplied with the development environment *IBS PC COP SWT (TDOS-PRO)*. Take care during the installation that the assigment of the adapter cable is correct (see Section 3).

4.6 Installation of the Device Driver

4 parameters are important when the device driver is installed.

- Base address in the I/O area of the PC (IO address)
- Board number (board no.)
- Base address of the 4 Kbyte MPM window in the memory area of the PC (MPM address)
- Interrupt number

The individual parameters have the following meaning:

Base address in the I/O area of the PC (I/O address)

This parameter stands for the address under which the controller board is addressed in the I/O address area of the PC. The PC I/O address is set with DIP switches on the controller board (see Section 4.1.1). The default value for the PC IO address is 120_{hex}. Refer to Section 4.1.1 for notes on alternative I/O addresses.

Board number (board no.)

The board number specifies for which controller board the device driver is to be loaded. The default value is 1, i.e. if the driver for the controller board no. 1 (can be set with DIP switches, see Section 4.1.2) is to be loaded, you do not need to specify this parameter. Permissible values for the board number are 1, 2, 3 and 4.

Memory address for the 4 Kbyte MPM window (MPM address)

Access to the Multi-Port Memory (MPM) is from the host PC via a 4 Kbyte MPM in the memory area below 1MB. The parameter *MPM Address* stands for the base address of this MPM window.

The board occupies an address area of 4 Kbytes from this base address onwards. Ensure that this area is not already used by other boards. An automatic check does **not** take place.

Table 4-15: Typical memory mapping of a standard PC

Memory	Address area			Used by
64 Kbytes	$F0000_{hex}$	to	$FFFFF_{hex}$	ROM BIOS
64 Kbytes	$E0000_{hex}$	to	$EFFFF_{hex}$	Expansions
64 Kbytes	$D0000_{hex}$	to	$DFFFF_{hex}$	EMS window
32 Kbytes	$C8000_{hex}$	to	$CFFFF_{hex}$	SCSI controller or LAN BIOS
32 Kbytes	$C0000_{hex}$	to	$C7FFF_{hex}$	EGA/VGA BIOS
32 Kbytes	$B8000_{hex}$	to	$BFFFF_{hex}$	Video memory (e.g. VGA text or EGA)
32 Kbytes	$B0000_{hex}$	to	$B7FFF_{hex}$	Video memory (e.g. monochrome or EGA)
64 Kbytes	$A0000_{hex}$	to	$AFFFF_{hex}$	Video memory (e.g. VGA graphics or EGA)
640 Kbytes	00000_{hex}	to	$9FFFF_{hex}$	Conventional memory

The base address of the MPM window may be placed in the area between 80000_{hex} to $FF000_{hex}$. As in practice this memory area already is occupied to a large extent (BIOS etc.), the address area that is available is normally limited to parts of the address segments D and E (addresses from $D0000_{hex}$ to $EFFFF_{hex}$). The address window should as far as possible be placed at the beginning or the end of the address area available in order not to split it up unnecessarily. The default value is $D0000_{hex}$.

Interrupt number

Every controller board uses a hardware interrupt for the communication with its device driver. This hardware interrupt is specified when the respective device driver is started.

The controller boards support the interrupts IRQ 3, 5, 7, 9, 10, 11, 12 and 15.

When several controller boards are used in one host PC, each installed controller board must use a different interrupt. On a standard PC, the interrupts IRQ 10, 11, 12 and 15 are mostly not used and, therefore, can be used by the device driver. The other interrupts are often used by standard PC components (serial ports COM1 and COM2, network adapters, etc.) so that they should not be used by controller boards.

Ensure that the interrupt has not been assigned to other components of your PC.

If you do not enter a parameter, a default is automatically used for this parameter.

4.6.1 Device Driver under MS-DOS

The device driver for the controller board is implemented under DOS as a TSR program (**T**erminate and **S**tay **R**esident), i.e. as a resident program running in the background.

- The TSR program for the host PC is called *IBSPCCB.EXE*.
- The TSR program for the coprocessor board (COP) is called *IBSCOP.EXE*.

Device driver on the host PC

Call *IBSPCCB* to start the device driver on the host PC. To call the device driver, change from the MS-DOS command line into the directory where the device driver is located, or specify the complete path with the call.

Device driver on the coprocessor board

- Transfer the device driver *IBSCOP.EXE* with the development environment *TDOS-PRO* to the *COP386* of the *IBS PC CB/COP/I-T*. Refer to the development environment manual for a precise description.
- On delivery of the *IBS PC CB/RTX486/I-T*, the device driver *IBSCOP.EXE* is located on drive A:\ (flash EPROM) of the *COP486* and is automatically started via an *AUTOEXEC.BAT* entry.

The device driver can (e.g. under MS-DOS 5.0) be loaded into the memory area above 640 Kbytes. In that case it will not occupy any memory space below 640 Kbytes. Refer to your DOS manual for the installation of drivers in the high memory area.

During the call you may also supply various parameters to the device driver. Calling *IBSPCCB /?* or *IBSPCCB /HELP* displays a list of the possible parameters.

Table 4-16: Examples for parameters supplied when calling the device driver

Call	Effect
IBSPCCB IO=100	Sets the base address in the I/O area of the PC to 100_{hex}
IBSPCCB BN=2	Sets the board number to *2*
IBSPCCB MPM=C800	Sets the base address of the MPM window to $C8000_{hex}$
IBSPCCB IRQ=10	Sets the interrupt number to *10*
IBSPCCB IO=100 BN=2	Sets the base address in the I/O area of the PC to 100_{hex} and the board number to *2*

A call with *IBSPCCB /INFO* allows you to check the set parameters of all device drivers.

If you call *IBSPCCB* without specifying further parameters, the following defaults are used:

- Base address in the IO area of the PC (I/O address): 120_{hex}
- Board number (board no.): 1
- Base address of the 4Kbyte MPM window in the memory area of the PC (MPM address): $D0000_{hex}$
- Interrupt number with driver software version ≤ 0.9: IRQ3
- Interrupt number with driver software version ≥ 0.91: IRQ15

When calling the device driver, you only need to specify the values which you want to change. The order of parameters is irrelevant. Entering *IBSPCCB / UNINSTALL* de-installs all device drivers.

4.6.1.1 Installation Assistance for DOS

The device driver *IBSPCCB.EXE* checks during its installation whether there is a controller board at the specified I/O address. If no controller board is found at the I/O address, an error message will be output and the device driver installation will not take place.

Any attempt of a multiple installation of device drivers with identical parameters will also be aborted with an error message. Only the device driver installed first remains operative.

The device driver provides a simple means of checking whether the PC has been installed correctly in the PC.

The device driver will not recognize any dual assignment of MPM addresses, I/O addresses or interrupts. Ensure that the set I/O address area, the memory area for the 4 Kbyte MPM window and the interrupt for the controller board are not already used by other PC components!

When the memory manager *EMM386.EXE* (Microsoft) is used, a 4 Kbyte memory area in the host PC must be protected from access by the memory manager. Therefore, make an addition to the following line in your host PC's *CONFIG.SYS*:

```
DEVICE = C:\DOS\EMM386.EXE     x=D000-D100
```

In this example, the entry in bold type protects a 4 Kbyte memory area from address $D0000_{hex}$ onwards from access by *EMM386.EXE*. Refer to your DOS manual for a more detailed description of the memory manager *EMM386*.

Ensure also that the use of SHADOW RAM or video cache does not conflict with the 4 Kbyte MPM window. In the event of a conflict, disable these options in the BIOS setup of your PC with the following settings:

SHADOW RAM	disabled
VIDEO CACHE	disabled

Do not install or de-install the device driver *IBSPCCB.EXE* in a Microsoft Windows® DOS box!

4.6.2 Device Driver Under Microsoft Windows

The driver software for Microsoft Windows® takes the form of a **D**ynamic **L**ink **L**ibrary (DLL, file name IBSPCCB.DLL). This DLL contains the Device Driver Interface and device drivers for four controller boards. Using the usual Windows copy procedure, copy the file *IBSPCCB.DLL* into the directory where your application program is located, or into the Windows root directory.

The use of additional drivers (such as calls of TSR programs under DOS or entry of the OS/2 device driver under IBM OS/2®) is not required under Microsoft Windows®!

The Windows root directory must also contain the file IBSPCCB.INI, which is used for parameterization. Enter the parameters required for initializing the controller board (I/O address, MPM address and interrupt number) into file *IBSPCCB.INI*. The following example shows the entry for the operation of a controller board.

```
[GENERAL]
EnableInitErrorMessage=TRUE

[BOARD1]
BoardInUseFlag=TRUE
IOAddress=120
MPMAddress=D000
IRQ=10

[BOARD2]
BoardInUseFlag=FALSE
IOAddress=120
MPMAddress=D100
IRQ=11

[BOARD3]
BoardInUseFlag=FALSE
IOAddress=120
MPMAddress=D200
IRQ=12

[BOARD4]
BoardInUseFlag=FALSE
IOAddress=120
MPMAddress=D300
IRQ=15
```

Figure 4-12: Example of entries in the *IBSPCCB.INI* file

4.6.2.1 Installation Assistance for Microsoft Windows

For instance, set for board 1 the entry *BoardInUseFlag,* to *TRUE*, to ensure that the controller board no. 1 is recognized during the initialization phase. Otherwise the conroller board will be identified as non-existent and will not be initialized - even if it does exist.

If you entered an invalid value, the initialization of the DLL will **not** be aborted.

- In the case of the entry *EnableInitErrorMessage=TRUE*, an error message will be output in a Windows message box when the DLL is loaded (start of the application program).
- In the case of the entry *EnableInitErrorMessage=FALSE*, **no** error message will be output when the DLL is loaded (start of the application program). In this case you can recognize only in the case of the error messages or DDI functions (e.g. when a data channel is opened for the first time) that an error has occurred during the initialization.

The parameters are identical with those of the driver software for DOS (see Page 4-17) - with the restriction that the values for *MPMAddress* may only be in the range from $A0000_{hex}$ to $FF000_{hex}$.

4.6.3 Device Driver Under IBM OS/2

For IBM OS/2® the device driver for the controller board takes the form of an OS/2 device driver.

An OS/2 device driver must be installed for each controller board!

The OS/2 device drivers for the controller board must be loaded when the host PC boots. Enter an OS/2 device driver for each controller board in the *CONFIG.SYS* file of your host.

The OS/2 device driver for the controller board is called *OS2_IBS.DRV*. Enter the complete path if the driver is not in the OS/2 root directory.

If you enter *OS2_IBS.DRV* without further parameters, the following defaults will be used:

- Base address in the I/O area of the PC (I/O address): 120_{hex}
- Board number (board no.): 1
- Base address of the 4Kbyte MPM window in the PC memory area (MPM address): $D0000_{hex}$
- Interrupt number: IRQ15

Table 4-17: Examples for entering parameters

Entry	Effect
Device=OS2_IBS.DRV IO=100	Sets the base address to 100_{hex}
Device=OS2_IBS.DRV IBSPCCB BN=2	Sets the board number to *2*
Device=OS2_IBS.DRV IBSPCCB MPM=C800	Sets the MPM address to $C8000_{hex}$
Device=OS2_IBS.DRV IBSPCCB IRQ=10	Sets the interrupt number to *10*
Device=OS2_IBS.DRV IBSPCCB IO=100 BN=2	Sets the base address to 100_{hex} and the board number to *2*

The order of parameters is irrelevant.

If you do not enter a parameter, the default parameter will be automatically used.

4.6.3.1 Installation Assistance for OS/2

If the specified PC I/O address does not match the address set by means of DIP switches on the controller board, the board initialization will be aborted and an error message will be output.

The controller board occupies an address area of 4 Kbytes. Ensure that this area is not already used by other PC boards (e.g. network adapters). A check does not take place.

Interrupt number

On the ISA bus, OS/2 does not permit that the same interrupt is used by more than one board. Take care that the interrupt used has not already been assigned. Otherwise, the initialization procedure will be aborted and an error message (Error at SetIRQ) will be output. The default value for the interrupt is 15.

4.7 Installation of the I/O Periphery

Please refer to the IBS installation manual *IBS SYS INST UM E* (Order No. 27 54 80 4). The manual informs you in several sections on the following subjects:

- System overview
- Installation of the I/O components
- Cabling recommendations
- Startup and function test
- Error correction
- Replacement of IBS components
- Cable diagrams

In addition, data sheets for all new Phoenix Contact IBS devices are available on request.

4.8 Software Tools for Startup

Various software tools allowing the access to your bus configuration without extensive programming efforts are available for the first startup.

- The process data monitor program *PCCBMONI.EXE* (supplied with the driver software)
- The diagnostic and configuration software *IBS SYS SWT* under DOS (Order No. 27 80 88 1)
- The operating software *IBS CMD SWT*, which allows configuring, monitoring and diagnostics of your InterBus-S system under Microsoft Windows®.

The software tools allow the setting of outputs. Before setting outputs ensure that this cannot lead to damage to property or injury to persons in connection with dangerous process peripherals (e.g. , unprotected shafts, motors, presses, etc.)

Refer to Section 7 of the respective manual for further information on the software tools.

4.9 Startup with Process Data Monitor Program

The driver software for IBS PC CB/.../I-T comes with a process data monitor program. The program provides the following services:

- First startup of an InterBus-S systems without much programming
- Test of your bus configuration
- Check of the connected configuration
- Display of the status of individual inputs (binary)
- Display of the status of input words (hexadecimal)
- Setting of individual outputs (binary)
- Setting of output words (hexadecimal)
- InterBus-S system reset

The functionality of this program includes also special functions such as generating a data file with all important data on the modules connected to the controller board.

Following the installation of your controller board and of the driver software, transfer the file *PCCBMONI.EXE* from the driver diskette to your hard disk. Call the process data monitor program by calling *PCCBMONI* . The following screen will appear:

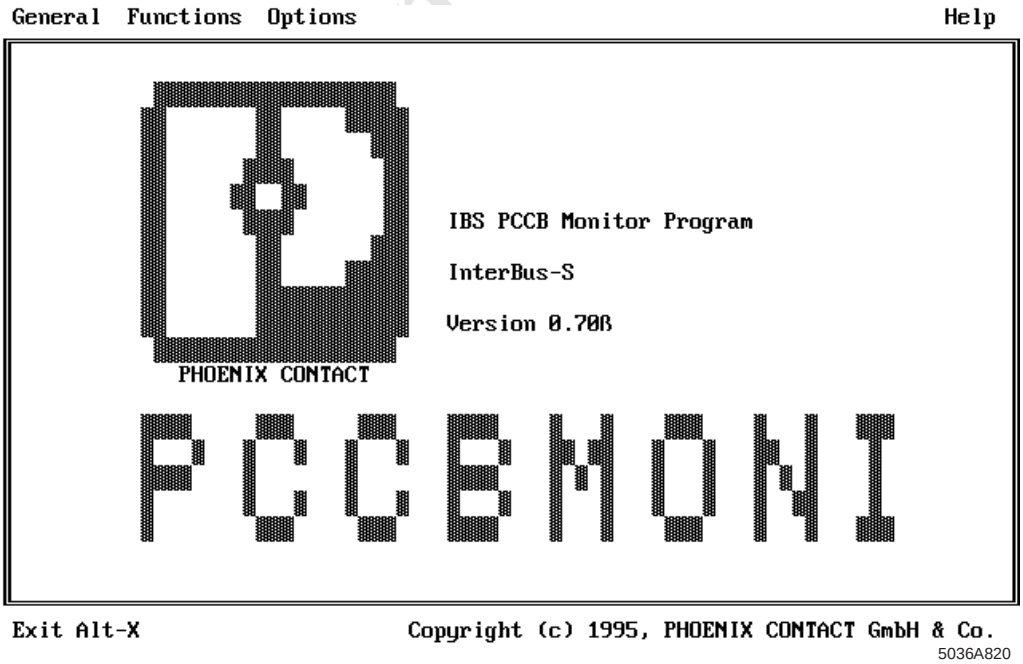


Figure 4-13: Main menu of the monitor program

You may specify the following parameters when calling *PCCBMONI.EXE* :

/MONO	Monochrome display
/NOTCOMPATIBLE	When your graphics adapter has cursor control problems (e.g. there is no cursor.)
/WARMSTART	The controller board initiates a warm start prior to every InterBus-S start.

/SWITCH	Indicates the current DIP switch setting. The DIP switch settings are only read in when the controller board is booted. After the setting has been changed, carry out a controller reset to make the change effective.
/HELP or /?	Information on command line options
/BOARDNO = n	Selection of the controller board which is to be accessed by the monitor program. Valid values for BOARDNO are 1, 2, 3 or 4.

The process data monitor program provides various menu items in three pull-down menus.

General: General program functions
Functions: IBS control and IBS data display functions
Options: Various program and bus functions

Press **[F1]** or the key combination **[ALT] [H]** to obtain help for the menu items.

4.9.1 The *Functions* Pull-Down Menu

Press **[F1]** or **[ALT] [F]** to access the *Functions* pull-down menu. It provides functions for InterBus-S control and data display.

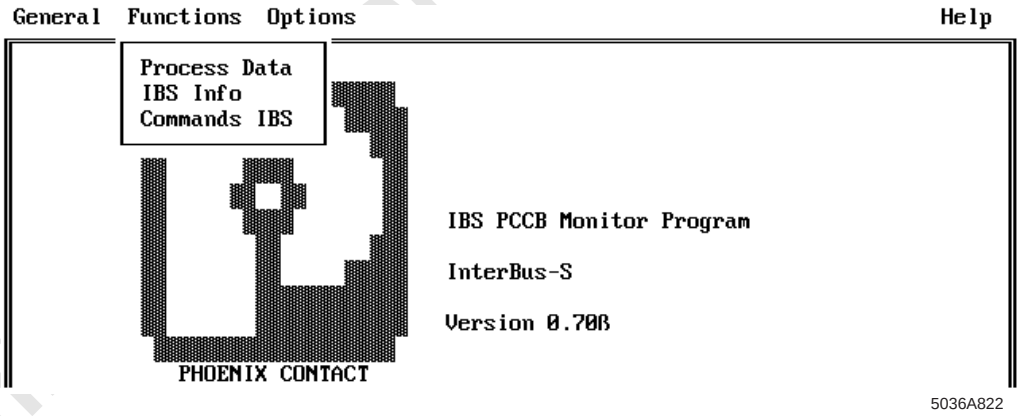


Figure 4-14: The *Functions* pull-down menu of the monitor program

Process Data	Indicates the input data and sets output data.
IBS Info	Indicates the bus status and bus errors.
IBS Commands	Makes it possible (from version 0.70 onwards) to issue commands, and indicates received messages

Process Data

Pressing **[P]** for *Process Data* will access the monitor mask.

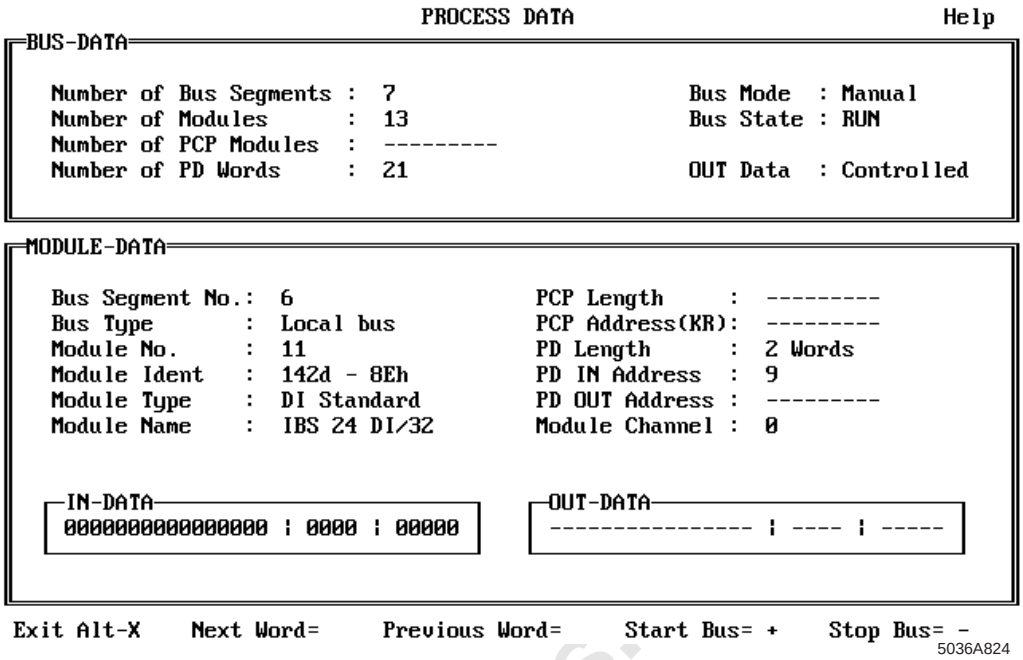


Figure 4-15: Monitor mask of the monitor program

Now press **[+]** to start InterBus-S data transmission. The *Bus State* display in the *BUS DATA* changes from *STOP* (red) to *RUN* (green). If there is no error, you can select all connected InterBus-S modules and check or change their I/O states.

Key assignment

[+]	Start of the data transmission on InterBus-S
[–]	Stop of the data transmission on InterBus-S
[↑]	Next process data word
[↓]	Previous process data word
[Page Up]	Jump to the next bus segment
[Page Down]	Jump to the previous bus segment
[Home]	Jump to the physically first InterBus-S module
[End]	Jump to the physically last InterBus-S module
[ESC]	Return to the start mask (bus cycles are not terminated)
[ALT] [X]	Monitor program termination (bus cycles are automatically stopped.)

The upper window shows the general system data on the left:

- Number of connected bus segments
- Number of connected modules (IBS devices)
- Number of connected PCP modules (IBS devices capable of communication)
- Number of process data words of the connected bus configuration

On its right, the upper window provides information on program settings and whether the bus is transmitting process data.

The lower window provides information on the currently selected module:

Bus Segment No.: Physical number of the bus segment where the current module is located (starting with 0)

Bus Type: (*Local Bus*, *Remote Bus*)

Module No.: Physical number of the module in the bus configuration (starting with 0)

Module Ident: Module ID code in decimal and hexadecimal notation

Module Type

Module Name

PCP Length: PCP address area occupied by the current module in the host

PCP Address: PCP address, corresponds to the communication reference (CR)

PD Length: Register length

PD IN Address: Physical address of the displayed input word

PD OUT Address: Physical address of the displayed output word

Module Channel: Channel number for modules with several process data words

IN-DATA: Current input word in binary, hexadecimal and decimal notation

OUT-DATA: Current output word in binary, hexadecimal and decimal notation

Error message When an error occurs in a bus configuration, a red window appears with an error message. The error message specifies the cause of the error (e.g. remote bus error 0 if the remote bus cable is not connected).

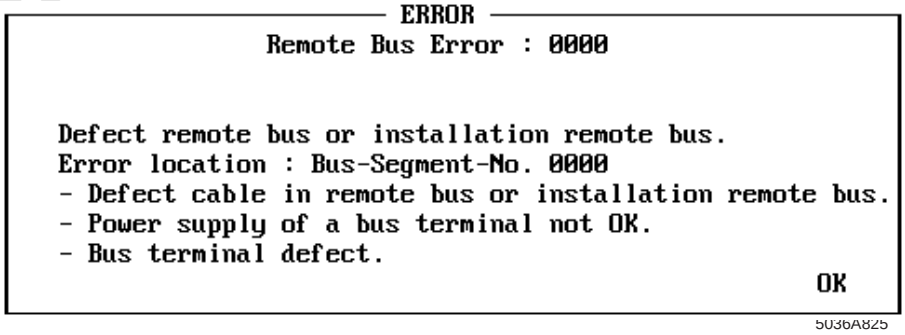


Figure 4-16: An error message in the monitor program

Refer to Section 10 for information on meaning, cause and remedy with the following error messages.

Info window The menu item *IBS-Info* calls the Info window, which provides an overview of the InterBus-S state and any error messages.

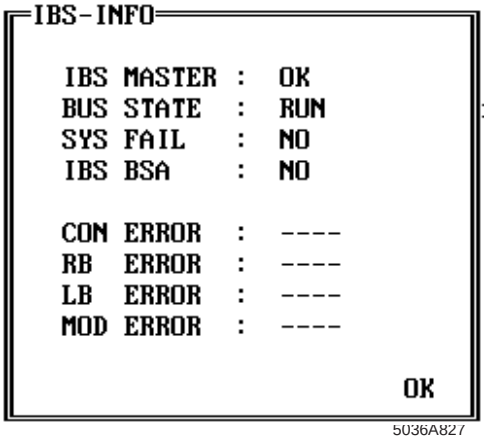


Figure 4-17: The monitor program Info window

CON ERR	Controller error
RB Error	Remote bus error
Local Bus Error	Local bus error (LB ERR)
Module Error	Module error (MOD ERR)

4.9.2 Issuing Commands with PCCBMONI

Commands

The menu item *Commands IBS* allows to manually issue IBS commands (requests) and to receive messages (confirmations) incl. parameters. You may also issue PCP commands and receive the corresponding messages. PCP commands are used, for example, for parameterizing intelligent IBS devices such as IBS V.24 or frequency inverters.

The menu item *IBS Commands* will only be supported from PCCPMONI Version 0.70 onwards.

The mask consists of two windows:

The left window is used for entering the desired commands (requests) with the appropriate parameters. Enter the command code in hexadecimal in the *REQ* line. If the command requires parameters, enter the number of parameters in the *PC* (Parameter Count) line, and the parameters themselves in the subsequent lines. Pressing **[ALT] [S]** sends the command to the controller.

Messages

The right-hand window displays the message sent last (Last Confirmation). The *CNF* line indicates the received message code. If the message has parameters, the *PC* (Parameter Counter) line indicates the number of subsequent parameters, and the following line contains the parameters themselves.

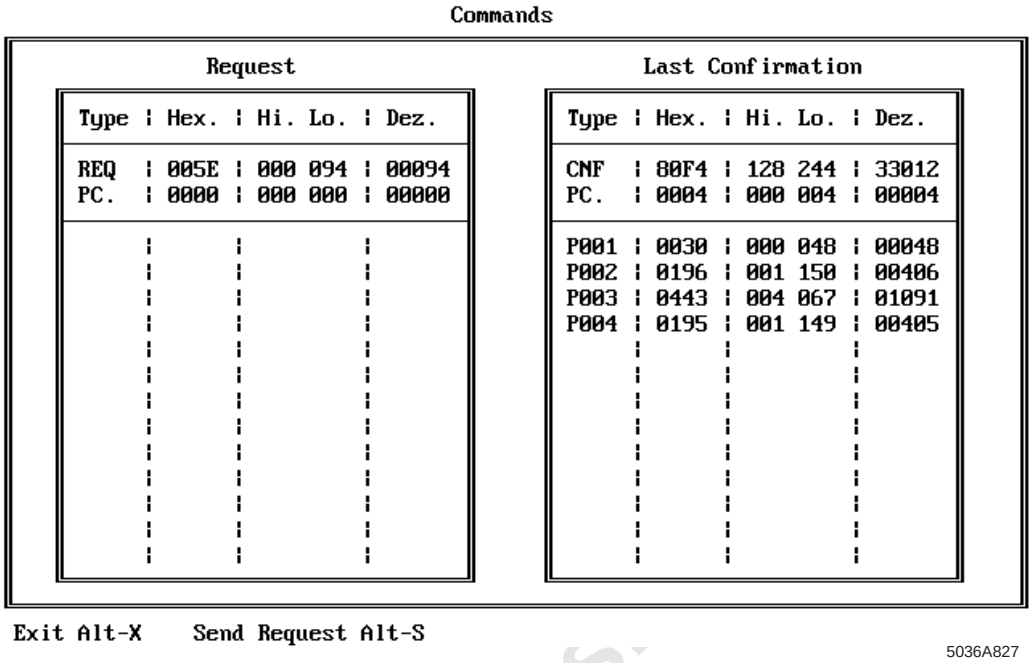


Figure 4-18: Mask for commands and messages

Three columns contain the individual parameters as follows:

Hex: Hexadecimal notation
Hi. Lo.: Decimal notation of the high (Hi.) and the low (Lo.) bytes. You can enter here, for example, the length code (hi.) and ID code (lo.) for an IBS device in decimal form.
Dec: Decimal notation

Use **[TAB]** or **[SHIFT] [TAB]** to change between the columns for the different types of representation. The conversion takes place automatically.

Scroll through the lines with [↑] or [↓]. When the number of parameters to be displayed in the right window exceeds the permissible range (number of parameters > 13), scroll through the parameter list with **[SHIFT]** [↑] or **[SHIFT**] [↓].

Using the following key combinations you can change between the windows of the *Functions* pull-down menu:

[ALT] [P] Change to the *Process Data* window
[ALT] [I] Change to the *Info* window
[ALT] [O] Change to the *Command* window

Using the keys [Esc], [X] or [Alt] [X] you can go to the next higher menu level.

4.9.3 The *Options* Pull-Down Menu

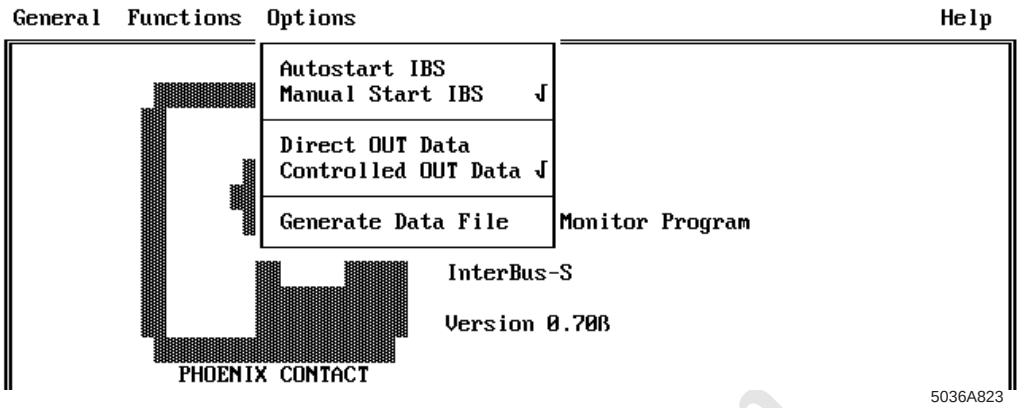


Figure 4-19: The *Options* pull-down menu

The pull-down menu *Options* is used to configure the monitor program.
Use the menu items *Autostart IBS* or *Manual Start IBS* to select whether, after an error has occurred, the controller board automatically attempts to restart the connected bus system (Autostart IBS). With *Manual Start IBS* the restart takes place only after [+] has been pressed.
Use the menu items *Direct OUT Data* or *Controlled OUT Data* whether the respective output is changed immediately after the entry of a value in the mask (Direct OUT Data) or after an acknowledgment with the [ENTER] key (Controlled OUT Data).
The menu item *Generate Data File* generates a file (IDLST.TXT) containing information on the currently connected bus configuration (length code, ID code, address, etc.).

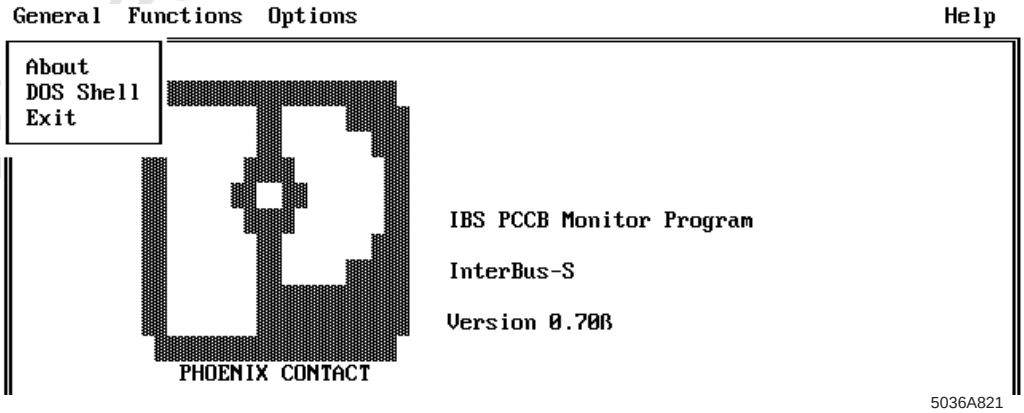


Figure 4-20: The *General* pull-down menu

About	Information on the monitor program version
DOS shell	Causes an entry to the DOS shell. InterBus-S retains the current state. Enter *EXIT* and [RETURN] to exit from the DOS shell.
Exit	Stops the data transfer to InterBus-S (Alarm_Stop_Request) and terminates the monitor program

Section 5

Interfaces Between Hardware and Software

This section provides information on

- the interfaces of the IBS controller boards;
- the general structure of the driver software.

5 Interfaces Between Hardware and Software

This section describes in general terms the interfaces between IBS controller boards and the driver software.

5.1 Multi-Port Memory

The central interface of the IBS controller boards is the **M**ulti-**P**ort **M**emory (MPM). The MPM is a memory on the motherboard which can be accessed by all MPM users (host, coprocessor board and IBS master board). The MPM users ("nodes") store all data intended for shared use. The MPM is the only connection between the MPM users.

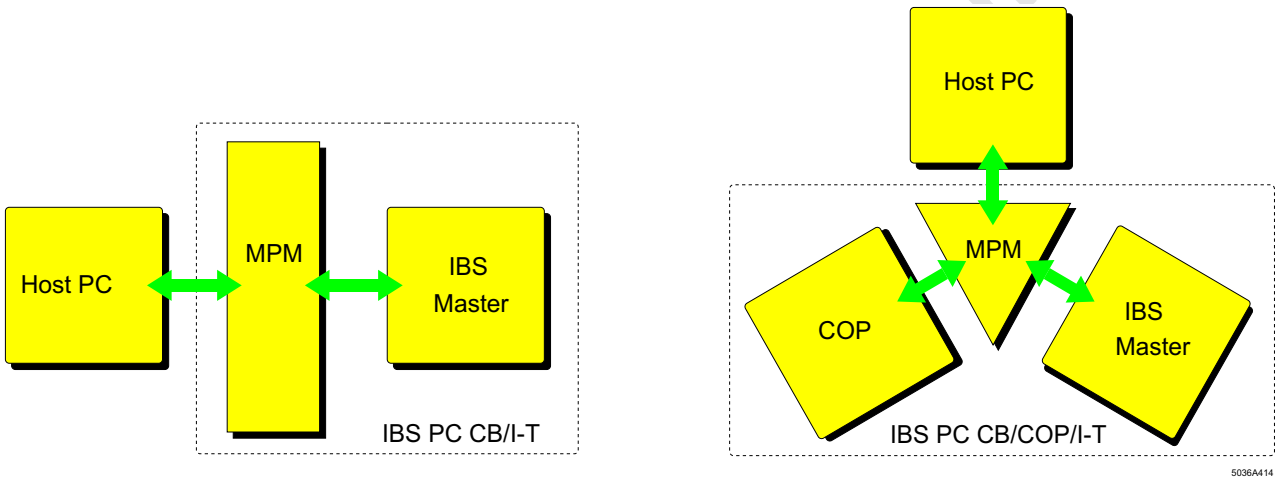


Figure 5-1: The MPM as the central interface of an IBS controller board

Figure 5-1 shows the central position of the MPM. The elements within the dashed lines are the hardware incorporated on the IBS controller boards.

A direct data exchange between the MPM users may only take place via the MPM. The MPM has a fixed structure which the user must not change.

The MPM may only be accessed via the device driver functions. Direct reading from or writing to the MPM is not allowed!

5.1.1 The MPM in the Host Address Area

The MPM is mapped to the address area of the host (between 640 KB and 1 MB). As the space available there is limited, only a memory window with a size of 4 Kbytes is used. The device driver shifts this window automatically (and fast, by means of the IBS controller board addressing logic programming) to the currently required address area of the MPM. The address of the window in the memory area of the host remains constant!

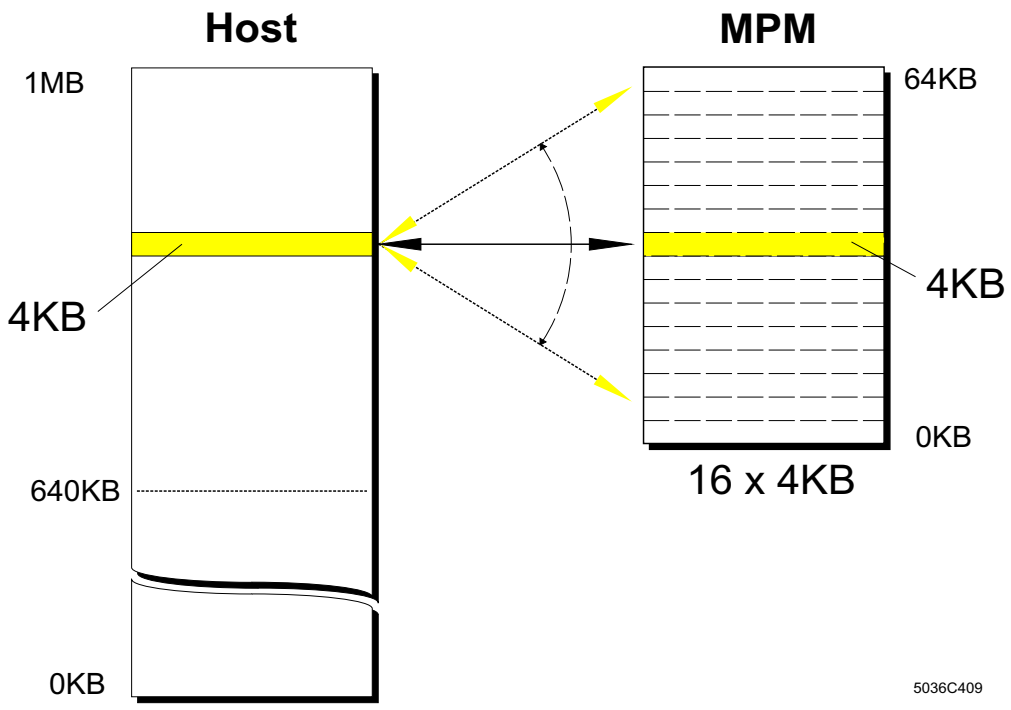


Figure 5-2: Shift of the MPM window by the device driver

5.1.2 Organization of the MPM

The 64 Kbyte MPM is divided into four user areas ("node areas"), each with 16 Kbytes. Three node areas are assigned to the MPM users ("nodes") host (PC), IBS master board (IBS MA) and coprocessor board (COP). The fourth node area is reserved for future expansions. Each MPM device may read from and write to its node area, but only read from the node areas of the others. When opening a data channel, the device driver automatically selects the correct node area on the basis of the device name. A 16 Kbyte node area consists of 4 DTI transfer areas, each with a size of 1 Kbyte for the data interface (DTI) and another 12 bytes used for the mailbox interface (MXI) and other purposes.

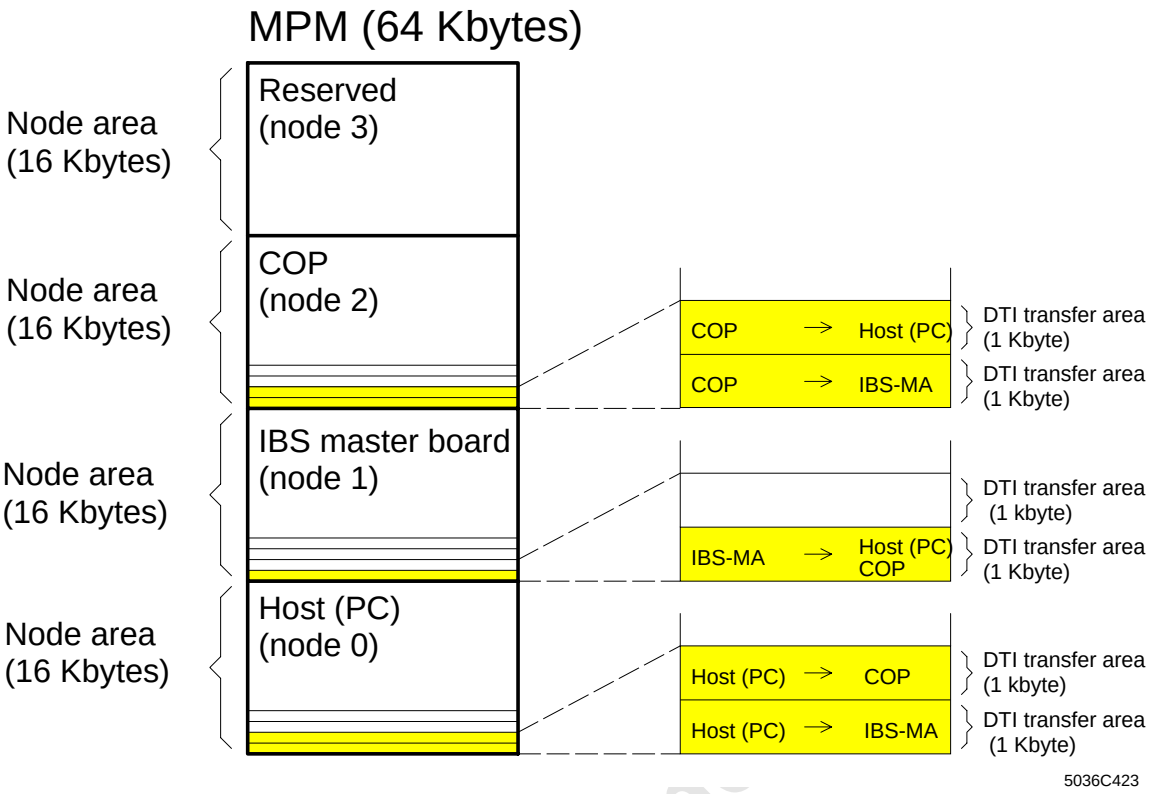


Figure 5-3: Organization of the MPM

The DTI address of a process data word within a node area consists of two components:

- the address offset of the DTI transfer area
- the relative address within this DTI transfer area.

Address offset: The address offset is the distance of a 1 Kbyte DTI transfer area from the beginning of the 16-byte node area.

Relative address: The relative address is the address of a process data word within this DTI transfer area. Like the *length* parameter it is specified in bytes, so that only even byte addresses are possible. Therefore, the first process data word is located at the relative address 0, the second one at the relative address 2, the third one at the relative address 4, etc.

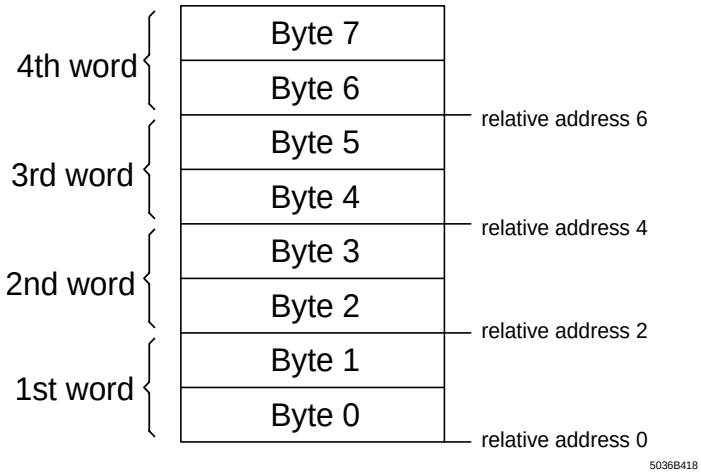


Figure 5-4: Relative addresses of process data words in a DTI transfer area

To access a process data word, the address offset and the relative address must be added up:

DTI address = address offset + relative address

The following offset constants are available to simplify the handling of the address offsets for the various data areas. They are defined in the *DDI_USR.H* include file:

Table 5-1: Offset constants

Access		Offset constant
from	to	
Host	IBS master board	DTA_OFFST_IBS_MASTER
COP	IBS master board	DTA_OFFST_IBS_MASTER
Host	COP	DTA_OFFST_PC_TO_COP
COP	Host	DTA_OFFST_COP_TO_PC

Therefore, you only need to add the offset constant corresponding to the access to the relative address.

DTI address = offset constant + relative address

When calling the functions DDI_DTI_WriteData and DDI_DTI_ReadData, enter the *DTI address* parameter (see the descriptions of these functions for further details).

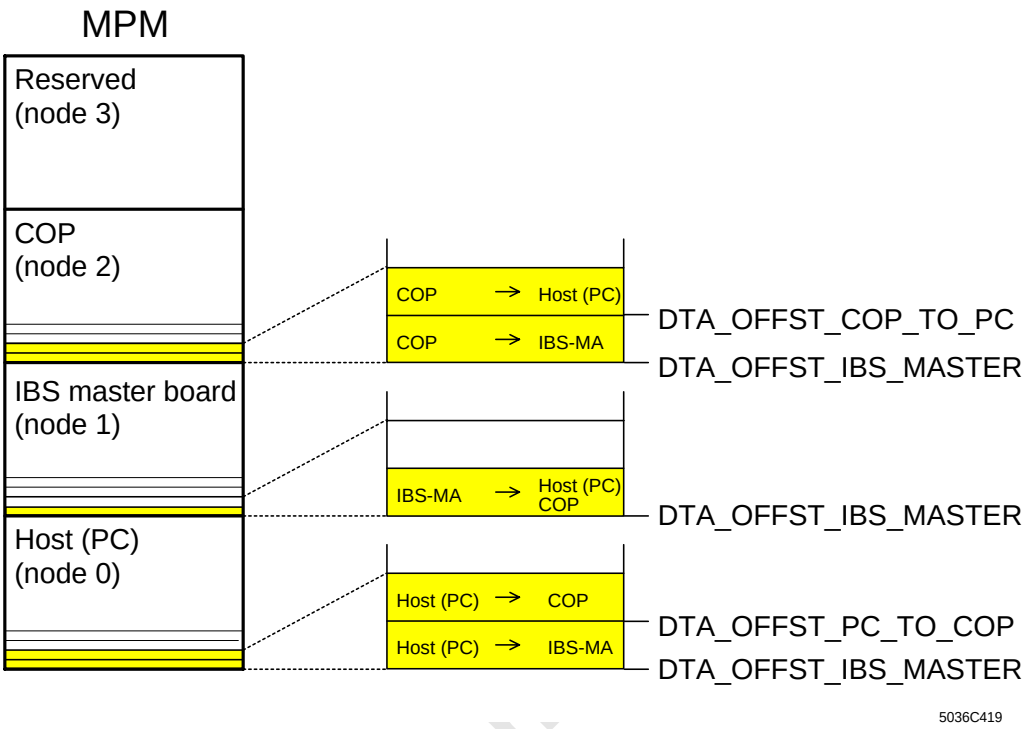


Figure 5-5: DTI transfer areas and offset constants

5.2 General Structure of the Driver Software

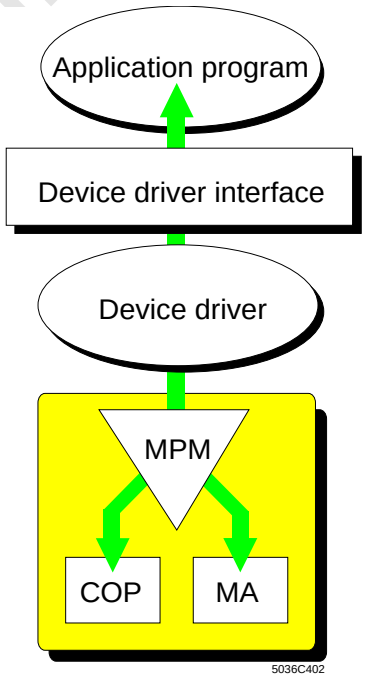


Figure 5-6: Structure of the driver software

Driver software for the controller boards is available for the operating systems DOS, Microsoft Windows® and IBM OS/2®.

The driver software consists of two parts:

1. The device driver Interface (DDI), a compiler-specific interface to the application program.
2. The operating-system-specific device driver (DD).
   The device driver links the host (PC) or the coprocessor board (COP), and the IBS master board (MA) via the MPM.

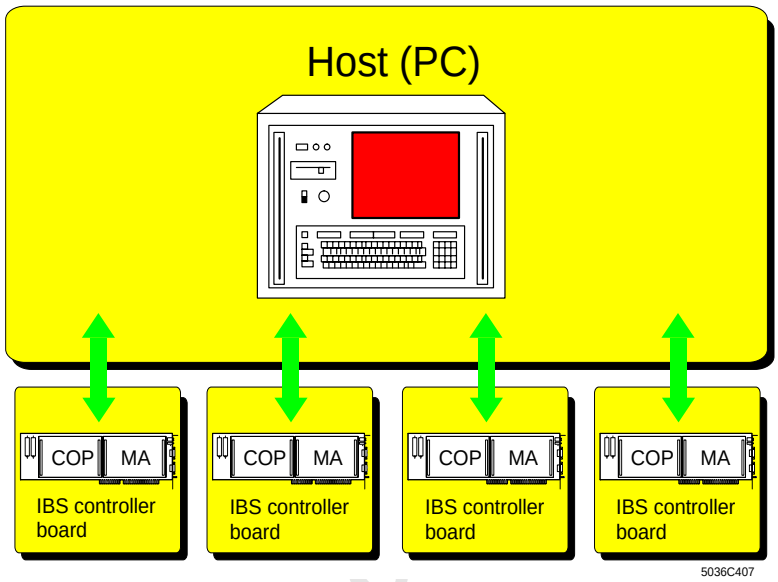


Figure 5-7: Operation of four IBS controller boards in one host

Up to four IBS controller boards can be installed in one host. One device driver must be installed for each IBS controller board. The device driver interface carries out the management and control of all device drivers:

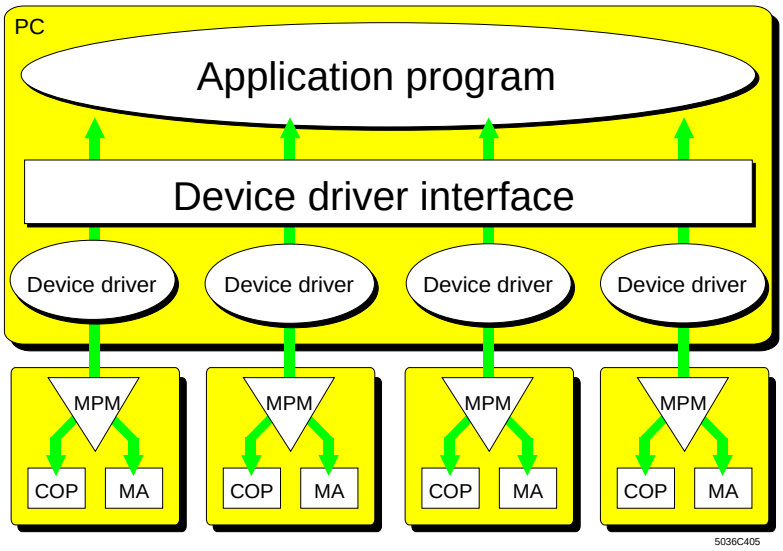


Figure 5-8: Control of four device drivers by the device driver interface

5.2.1 Implementation of the DDI and the DD

Please refer to the driver software manual (IBS PC CB SWD UM E, Order No. 27 53 96 0) for the implementation of the device driver interface and of the device drivers for the various operating systems and the supported compilers.

5.2.2 Structure of the Driver Software on the Coprocessor Board

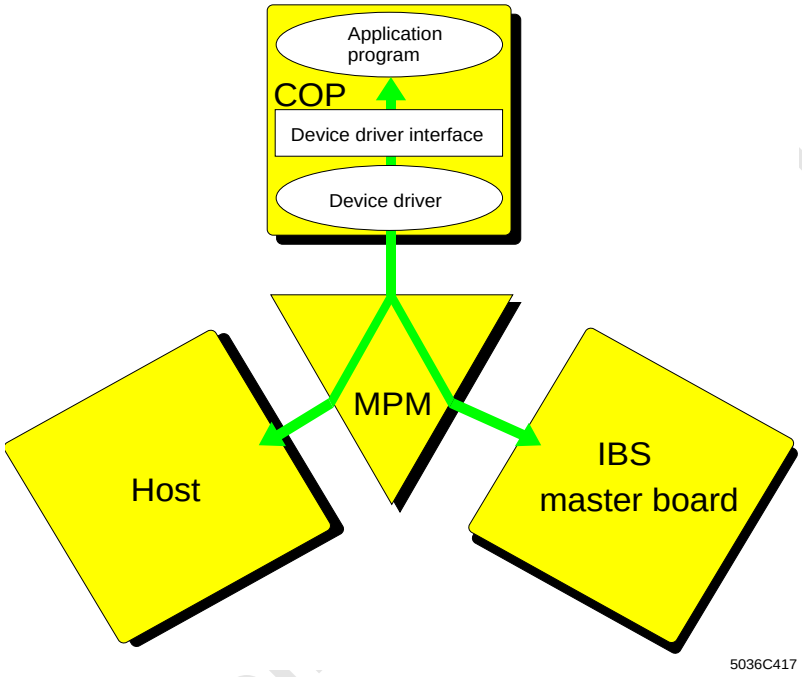


Figure 5-9: Structure of the driver software on the coprocessor board (COP)

Coprocessor programming does not differ from host programming under DOS. A DOS program for the host may also be run on the coprocessor board. Call the TSR program IBSCOP.EXE instead of IBSPCCB.EXE as device driver for the coprocessor board.

- Transfer the device driver *IBSCOP.EXE* with the development environment-*TDOS-PRO* to the *COP386* of the *IBS PC CB/COP/I-T*. Refer to the development environment manual for the precise description.
- Upon delivery of the *IBS PC CB/RTX486/I-T*, the device driver already is on drive A:\ (flash EPROM) of the *COP486* and is automatically started by means of an *AUTOEXEC.BAT* entry.

Note that the coprocessor board has by nature no keyboard, monitor and hard disk of its own. The use of the host's keyboard and monitor (terminal mode) as well as its hard disk for the coprocessor system is described in the documentation of its operating systems (TDOS/RTXDOS).

5.2.3 Explanation of Driver Software Terms

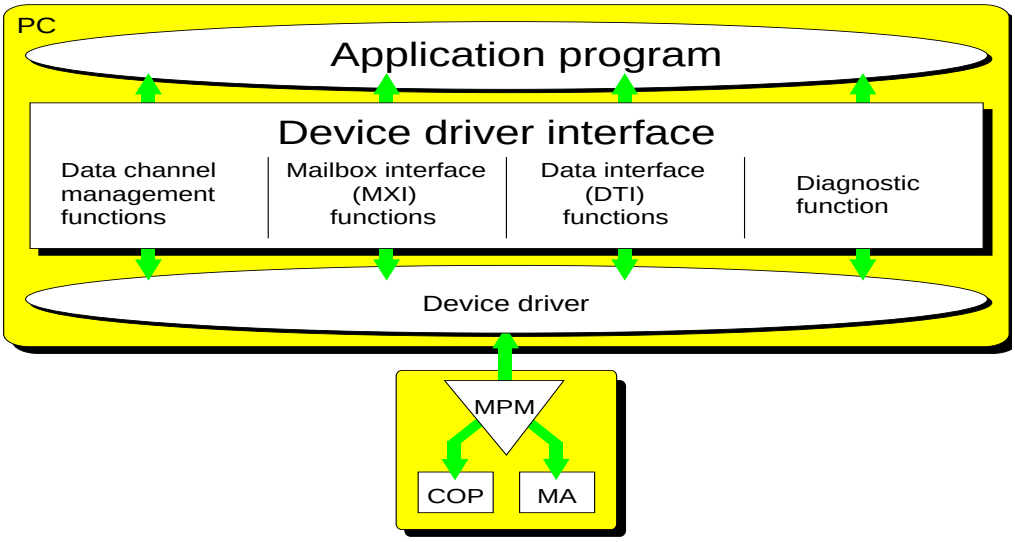


Figure 5-10: The DDI as interface to the device driver

5.2.3.1 Management of Data Channels

Node

A user of the MPM with the associated device driver is referred to as a node. The following nodes are used:

Node 0: Host with associated device driver
Node 1: IBS master board with associated device driver (the device driver of the IBS master board is an integral part of its firmware.)
Node 2: Coprocessor board with its associated device driver

Node handle

A node handle identifies an open data channel to a node.

Device name

Name of a device to which a data channel is to be opened. The name identifies the IBS controller board (board number 1 to 4) and the MPM user there (host, IBS master board or coprocessor board). See also the descriptions for the functions *DDI_DevOpenNode* and *DDI_DevCloseNode*.

See also the descriptions of the functions *DDI_DevOpenNode* and *DDI_DevCloseNode* in the driver software manual *IBS PC CB SWD UM E*.

For a simplified handling, a string allocating the board number and the MPM user for opening any of the available data channels is available for the *device name* parameter.

Table 5-2: Opening a data channel from the host to the IBS master board

Controller board (board number)	MPM node	Interface	String for the parameter *device name* (devName)
Board 1	IBS master board	Mailbox interface	IBB1N1_M
Board 1		Data interface	IBB1N1_D
Board 2		Mailbox interface	IBB2N1_M
Board 2		Data interface	IBB2N1_D
Board 3		Mailbox interface	IBB3N1_M
Board 3		Data interface	IBB3N1_D
Board 4		Mailbox interface	IBB4N1_M
Board 4		Data interface	IBB4N1_D

Table 5-3: Opening a data channel from the host to the coprocessor board (for IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T)

Controller board (board number)	MPM node	Interface	String for the parameter *device name* (devName)
Board 1	Coprocessor board	Mailbox interface	IBB1N2_M
Board 1		Data interface	IBB1N2_D
Board 2		Mailbox interface	IBB2N2_M
Board 2		Data interface	IBB2N2_D
Board 3		Mailbox interface	IBB3N2_M
Board 3		Data interface	IBB3N2_D
Board 4		Mailbox interface	IBB4N2_M
Board 4		Data interface	IBB4N2_D

Table 5-4: Opening a data channel from the coprocessor board to the host (for IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T)

Controller board (board name)	MPM node	Interface	String for the parameter *device name* (devName)
Controller board 1 to controller board 4	Host	Mailbox interface	IBB1N0_M
		Data interface	IBB1N0_D

Table 5-5: Opening a data channel from the coprocessor board to the IBS master board (for IBS PC CB/COP/I-T and IBS PC CB/RTX486/I-T)

Controller board	MPM node	Interface	String for the parameter *device name* (devName)
Controller board 1 to controller board 4	IBS master board	Mailbox interface	IBB1N1_M
		Data interface	IBB1N1_D

When using C, put the character strings for the *device name* parameter between double inverted commas (e.g. "IBB1N1_M"); when using *Turbo-Pascal*, put it between single inverted commas (e.g. ‘IBB1N1_M‘).

Data Consistency

The data consistency is the number of bytes which a node can read or write without another node accessing these bytes at the same time. This means that the access of a node to a number of bytes determined by the data consistency must have been completed before another node can access these bytes. Thus, the MPM logic prevents simultaneous accesses of the nodes to MPM data.

Possible values for data consistency are:

- Word data consistency (2 bytes) DTI_DATA_WORD
- Longword data consistency (4 bytes) DTI_DATA_LWORD
- Byte data consistency (1 byte) DTI_DATA_BYTE
- 48-bit data consistency (6 bytes) DTI_DATA_48 BIT

5.2.3.2 Mailbox Interface

The mailbox interface (MXI) is used to transfer messages between the MPM users.

Certain commands are acknowledged with a message indicating whether the command has been executed successfully or whether errors have occurred during the execution. The acknowledgment is also transferred via the mailbox interface.

If the application program fails to fetch a message, a time-out error is output after 8 minutes. This error message is also a message and overwrites the message entered before. If more than one message is supplied, the period before a time-out occurs is only 8.1 seconds.

5.2.3.3 Data Interface

The data interface (DTI) is used to transfer I/O data between the MPM nodes. The transfer takes place without an acknowledgment.

5.2.3.4 Diagnostic Function

In the MPM, the IBS controller board has two registers for evaluating error indications by the application program:

- The register for diagnostic bits contains information on the operating and diagnostic state of the IBS controller board.
- The register for diagnostic parameters provides additional information on the error type or the error location.

Each register occupies one word in the address area of the MPM. Both registers can be evaluated with the diagnostic function. The diagnostic function is described in detail in the driver software manual.

5.3 Use of the Static RAM

(for IBS PC CB/COP/I-T und IBS PC CB/RTX486/I-T)

The coprocessor board of the controller IBS PC CB/COP/I-T provides 128 Kbytes of static RAM (SRAM). The SRAM is backed up by the motherboard batteries. When the host is off or no IBS controller board is installed, the data stored in the SRAM is retained for up to 2.5 years.

SRAM on the COP386

Two functions are available for writing to and reading from the SRAM. Their syntax is similar to that of the data interface functions in the device driver interface.

Up to 64 Kbytes can be transferred per function call. When the amount of data you want to transfer is larger, call the function twice. Enter an offset of 64 Kbytes for the 2nd call. The include files and libraries (for C) and units (for Pascal) required for the use of the SRAM are described in the driver software manual.

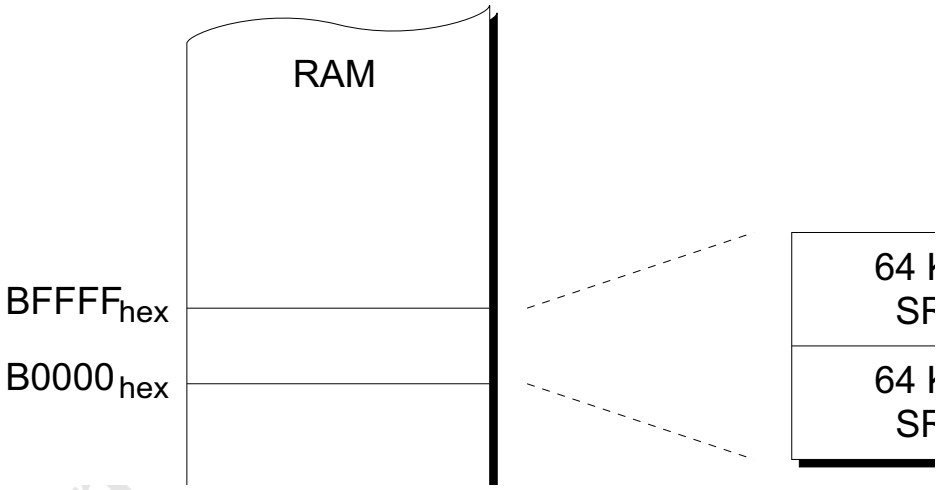


Figure 5-11: Segmentation of the SRAM

SRAM on the COP486

On the *COP 486*, *RTXDOS* allows to access the SRAM via normal file functions as drive D:\ .

5.4 Communication Between Host and COP

The communication between the host and the COP takes place basically in the same way as between the host and the IBS master board or the COP and the IBS master board. To open a data channel between host and COP the device name (devName) specified in Table 5-3.

5.4.1 Structure of a Message Between Host and COP

The commands and messages for the communication between host and COP are not predefined. You can define these commands/messages yourself, while keeping to a predefined structure. This structure is the same as the structure of the IBS master board control commands and messages.

Commands and messages without parameters contain only the actual command or message code. Further parameters do not follow.

When a command or message has parameters, the command or message code is followed directly by the parameter count (PC). The parameter count indicates the number of words that follow.

All message blocks are transferred and accepted in the so-called mailbox format. The structure of these blocks of the data type *USIGN16* is as follows:

Commands or messages without parameters

Word 1	*Command code$_{hex}$*		Word 1	*Message code$_{hex}$*	
Bit	15 8	7 0	Bit	15 8	7 0

Commands or messages with parameters

Word 1	*Command code$_{hex}$*		Word 1	*Message code$_{hex}$*	
Word 2	*Parameter count (n)*		Word 2	*Parameter count (n)*	
Word 3	*Parameter 1*		Word 3	*Parameter 1*	
Word4	*Parameter 2*		Word 4	*Parameter 2*	
	...			...	
Word n+2	*Parameter n*		Word n+2	*Parameter n*	
Bit	15 8	7 0	Bit	15 8	7 0

5.5 Monitoring by Watchdogs

The include files and libraries (for C) and the units (for Pascal) required for using the watchdogs are described in Section 4.6.

The host and the coprocessor board (COP) can be monitored each by its own watchdog circuit. The IBS master board (MA) is always monitored by a watchdog circuit.

5.5.1 IBS Master Board Watchdog

When the IBS master board watchdog trips, the IBS system is reset, all outputs of the IBS system are reset, and the master board's *SysFail* signal is set. The watchdog is enabled all the time and cannot be influenced by the user.

5.5.2 Watchdog for Host Monitoring

A watchdog circuit for monitoring your PC program (PC ”crash“, program ”hang-up“) is incorporated into the IBS controller board. When it trips, the watchdog places the IBS system in a defined condition (reset of all outputs).

The watchdog does not influence the host, which means that, for example, no host reset is carried out!

If you want to use the watchdog, enable it from the application program. As defaul it is disabled.

To enable the host watchdog, call the *EnableWatchdog ()* function. After enabling, the watchdog cannot be disabled by the software; it can only be deactivated by switching off the host or by a hardware reset.

The host watchdog monitoring period is permanently set to146 ms: Within this time the watchdog must be triggered by the *TriggerWatchDog ()* function in the application program, or else it will cause an IBS system reset.

Under Windows, the use of the watchdog for host monitoring is not recommended. It is not ensured that your program allows watchdog triggering within the predefined time. An example: You are shifting the frame of a window with your mouse. As long as you are holding the frame of the window with your mouse, your program will not continue to run!

5.5.3 Coprocessor Board Watchdog

The coprocessor board can also be monitored by its own watchdog circuit. The function and operation of the watchdog is identical with the PC monitoring watchdog, with the exception of the reset carried out when the watchdog trips.

To enable the coprocessor board watchdog, call the *EnableWatchDog ()* function. After enabling, the watchdog cannot be disabled by the software; it can only be deactivated by a hardware reset.

The coprocessor board watchdog is permanently set to 125 ms. Within this time the the watchdog must be triggered by calling the *TriggerWatchDog ()* in the application program, or else it will cause a coprocessor board reset.

The watchdog state can be read out with the *GetWatchDogState( )* function. Using this function you can, when starting the application program, determine whether the COP watchdog had triggered a coprocessor board reset. The *ClearWatchDog()* function resets the COP watchdog.

5.5.4 The *SysFail* Signal

For each node connected to the MPM, a separate area is reserved, which is used e.g. for status messages or check-back signals. One of these status signals is the *SysFail* (system failure) signal. It is set in the case of a system error of the respective user, e.g. when the watchdog has tripped. Using the *GetSysFailRegister* function you can read out the *SysFail* signal of any MPM node

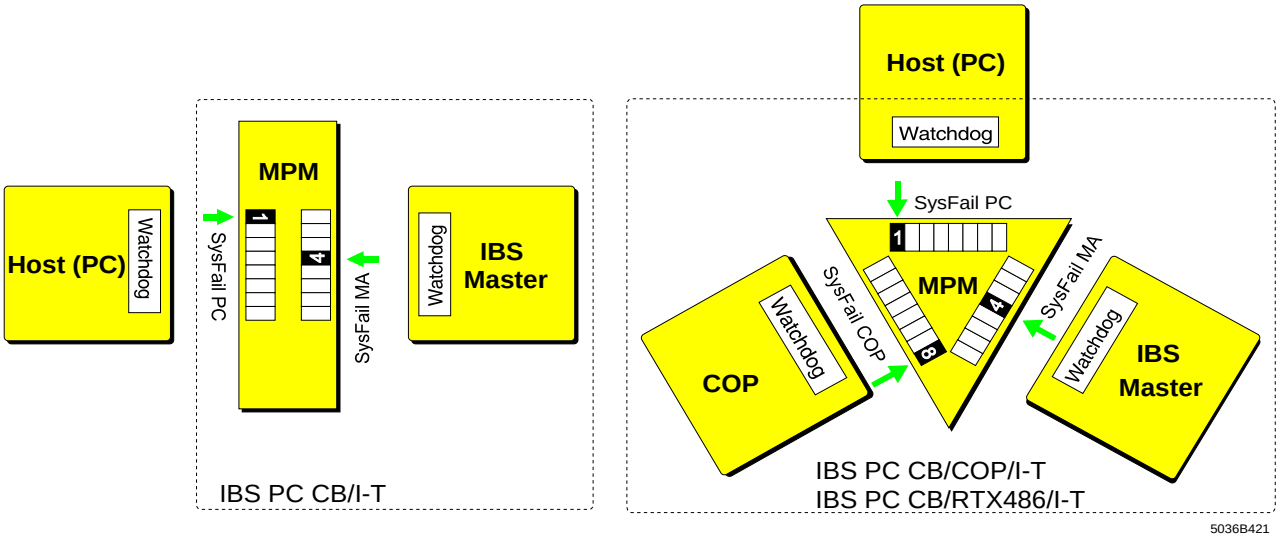


Figure 5-12: The *SysFail* signal in the MPM

5.6 Application Program Downloading to the COP

(only for IBS PC CB/COP/I-T)

The development environment IBS PC COP SWT (TDOS-PRO, Order No. 27 52 12 3) is available for a convenient handling of the coprocessor board during program development. By means of this development environment, the following operations can be implemented in a convenient way:

- Application program downloading
- Evaluation of an IBS controller board status report
- IBS controller board reset
- Flash EPROM programming
- Terminal function for the IBS controller board

IBS PC COP SWT comes complete with the manual for this development environment and the adapter cable for extending the serial interface of the COP (see Section 3).

Section 6

INTERBUS-S-specific Programming

This section shows on the basis of a bus configuration example the programming of typical InterBus-S functionalities such as

- physical addressing and logical addressing;
- group definition and disabling/reenabling groups.

6 InterBus-S-specific Programming

6.1 Identification of the Connected IBS Devices

The connected IBS devices are identified on the basis of three features:

- Physical position of the IBS device in the bus configuration
- Length code of the IBS device (describes the address space requirement in the host)
- Identification code (short: ID code) of the IBS device

A table (device list) describes the structure of the IBS system.

Length code	ID code	for device 1
Length code	ID code	for device 2
Length code	ID code	for device 3
...	...	
Length code	ID code	for device n

Bit | 15 ..... 8 | 7 ..... 0 |

Figure 6-1: Principle of the device list structure

Key:

Length code: The length code describes the address space requirements of the IBS device in the host.

ID code: Identification code (short: ID code) of the IBS device

The configuration of the IBS system is entered in the ID list. This requires that the length codes and the ID codes of the individual IBS devices are known (see the IBS device overview).

6.1.1 Physical Counting Mode for Bus Segments and IBS Devices

A bus segment consists of a remote bus device and the incoming remote bus cable. If this remote bus device is a bus terminal module, the devices of a local bus branching off from it are also part of its bus segment. Bus segment 0 (short: BS 0) is the bus segment whose bus terminal module is directly connected with the controller board in the host via the remote bus. Bus segment 1 is connected via the remote bus to the bus terminal module of bus segment 0, etc.

Bus terminal modules for the installation remote bus and installation remote bus devices have a bus segment number of their own
(see Figure 6-2, bus segments 2 to 5).

6.1.2 Bus Configuration Example

The following pages contain examples to describe the different types of addressing. The bus configuration shown is the same for all examples and consists of the following:

Table 6-1: IBS devices of the configuration example

Module no.	Bus segment	Module	Length code*	ID code*
1	BS 0	IBS ST 24 BK-T	00_{hex}	08_{hex}
2	BS 0	IB ST 24 DI 16/4	01_{hex}	BE_{hex}
3	BS 0	IB ST 24 AO 4/SF	04_{hex}	$7D_{hex}$
4	BS 0	IB ST 24 DO 16/3	01_{hex}	BD_{hex}
5	BS 0	IB ST 24 PT 100 4/4	04_{hex}	$7B_{hex}$
6	BS 1	IBS IP DIO 1/24-P	01_{hex}	07_{hex}
7	BS 2	IBS IP CBK 1/24-F	00_{hex}	$0C_{hex}$
8	BS 3	IBS IP CDI 1/24-F	81_{hex}	$0A_{hex}$
9	BS 4	IBS IP CDO 1/24-F	81_{hex}	09_{hex}
10	BS 5	IBS IP CDI 1/24-F	81_{hex}	$0A_{hex}$
11	BS 6	IBS 24 BK I/O-T	01_{hex}	$0B_{hex}$
12	BS 6	IBS 24 DI/32	02_{hex}	$8E_{hex}$
13	BS 6	IBS AI 3	04_{hex}	47_{hex}

* Refer to the IBS device overview for the length and ID codes for all Phoenix Contact IBS devices.

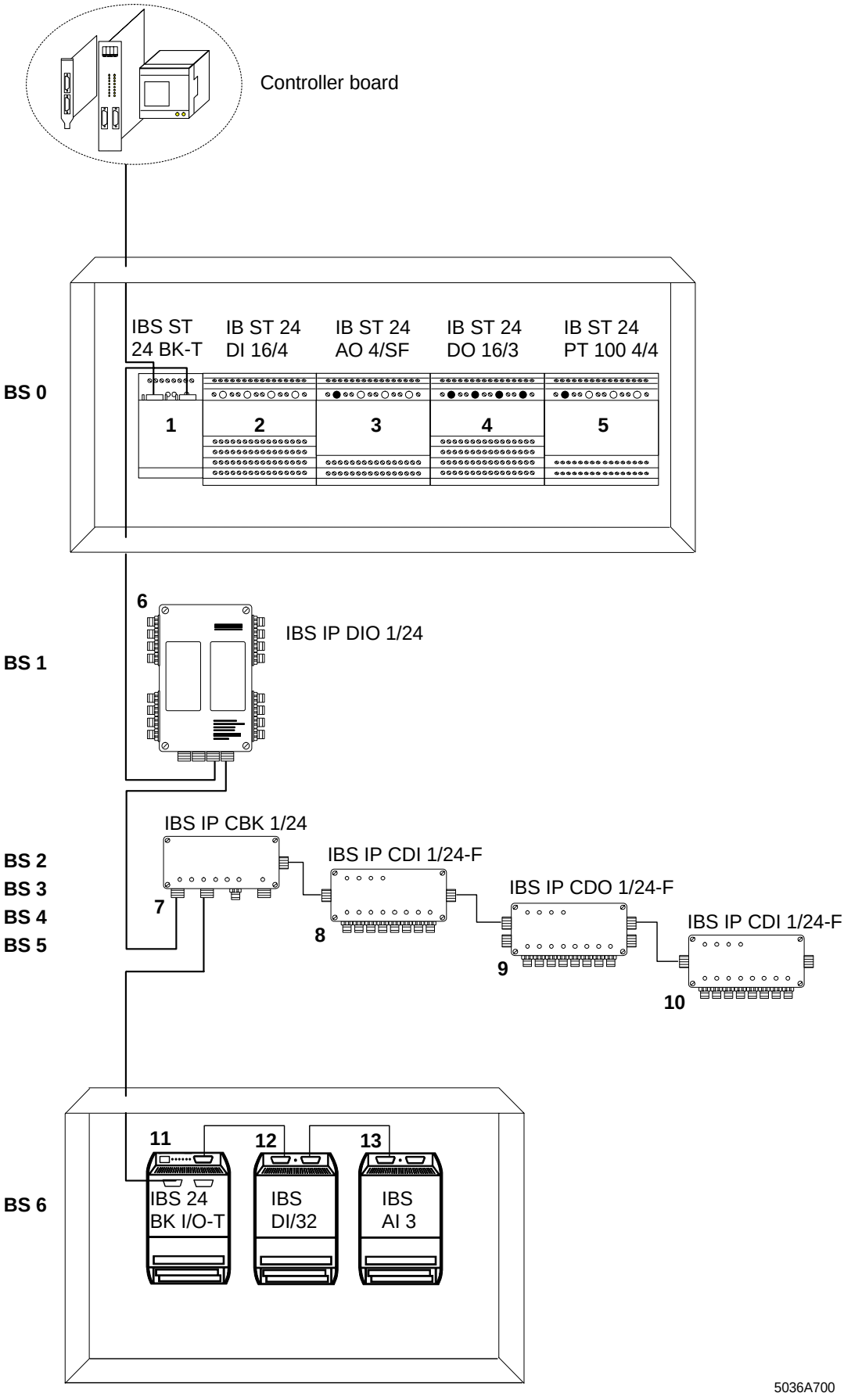


Figure 6-2: Bus configuration for the addressing examples

6.1.3 InterBus-S Addressing Modes

Two different addressing modes are possible.

Physical addressing

In this mode the controller board automatically determines the addresses of the IBS devices automatically on the basis of the physical order of IBS devices in the bus configuration.

Logical addressing

In this mode the programmer can freely choose the addresses for the IBS devices. The controller board then addresses the IBS devices as defined in the application program.

Logical addressing is useful in order to

- optimize the memory segmentation;
- avoid address shifts when your system is to be expanded;
- allow configuration changes without completely rearranging the addressing;
- improve the clarity of the system arrangement. The bus segments of a part of the system (e.g. of a switch cabinet, a machine) can be assigned to a defined group.

6.2 Physical Addressing of IBS Devices

With physical addressing, the controller board addresses the IBS devices automatically on the basis of the physical order of devices in the bus configuration. Under firmware 3.x the controller board reserves at least 1 word (16 bits) in the multi-port memory for each device that supplies or processes process data.

With physical addressing, the controller board automatically addresses the IBS devices anew after the bus configuration has been changed or expanded. Take appropriate steps in the application program (e.g. inquiry of the configuration), as address shifts may take place!

In contrast to physical addressing, logical addressing allows you to assign the addresses of the IBS devices in your system independent of the actual physical order of their arrangement in the bus system (see Section 7.2.2).

6.2.1 Addresses in the Physical Addressing Mode

6.2.1.1 Assignment of the Input Addresses by the Controller Board

Table 6-2: Assignment of the input words to the IBS devices

Module no.	Module	Input word (IW)
1	IBS ST 24 BK-T	—
2	IB ST 24 DI 16/4	IW 0
3	IB ST 24 AO 4/SF	—
4	IB ST 24 DO 16/3	—
5	IB ST 24 PT 100 4/4	IW 1, IW 2 , IW 3, IW 4
6	IBS IP DIO	IW 5
7	IBS IP CBK	—
8	IBS IP CDI	IW 6
9	IBS IP CDO	—
10	IBS IP CDI	IW 7
11	IBS 24 BK I/O-T	IW 8
12	IBS 24 DI/32	IW 9, IW 10
13	IBS AI 3	IW 11, IW 12, IW 13, IW 14

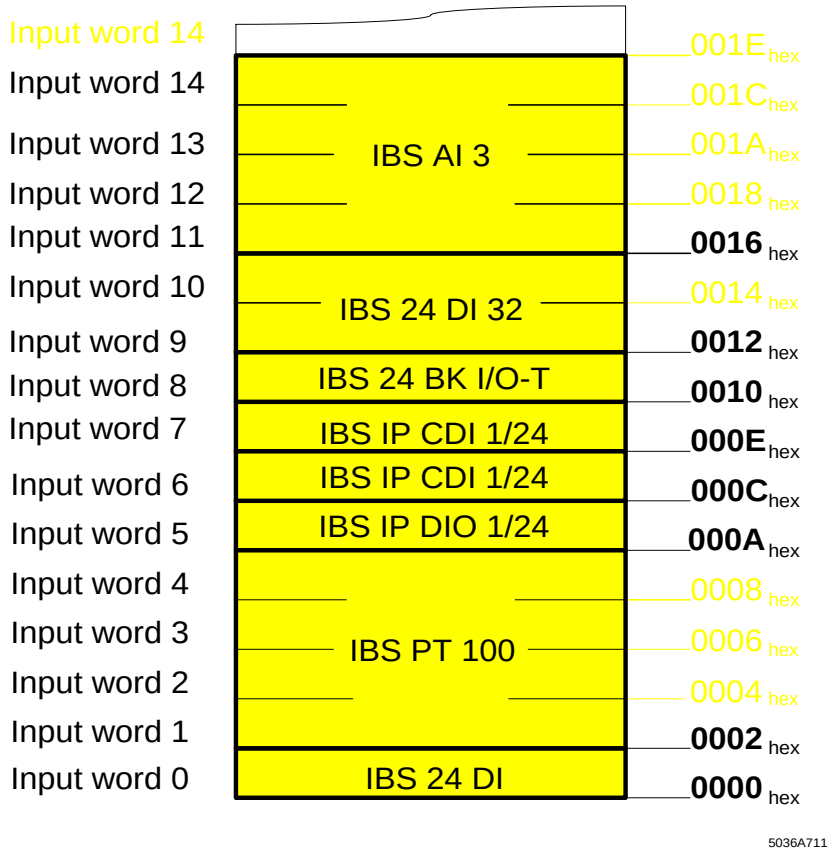


Figure 6-3: Input addresses in the memory map (IN buffer)
The start addresses of the modules are in bold type.

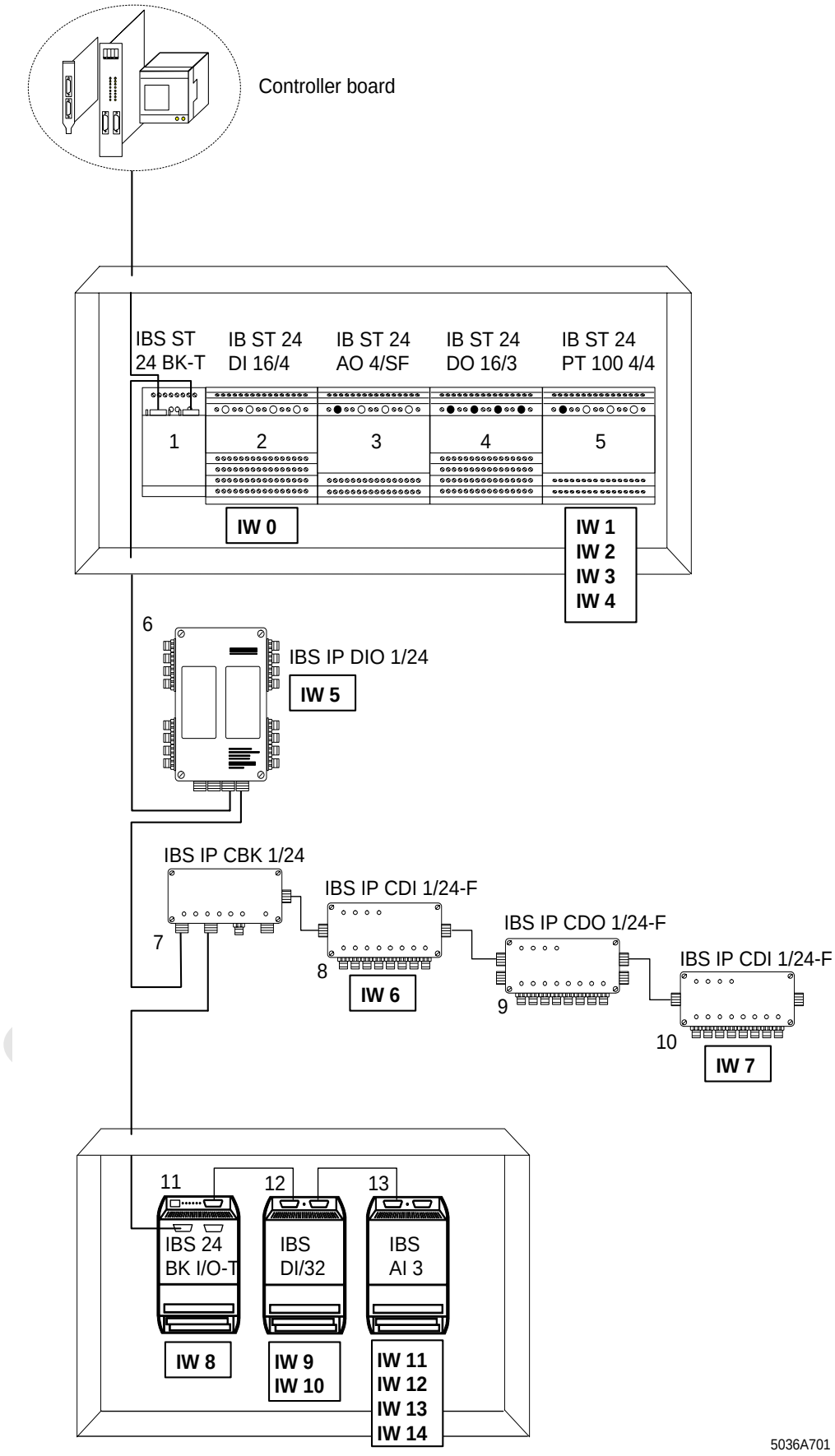


Figure 6-4: Input addresses under physical addressing

6.2.1.2 Assignment of the Output Addresses by the Controller Board

Table 6-3: Assignment of the output words to the IBS devices

Module no.	Module	Output word (OW)
1	IBS ST 24 BK-T	-
2	IB ST 24 DI 16/4	-
3	IB ST 24 AO 4/SF	OW 0, OW 1, OW 2, OW 3
4	IB ST 24 DO 16/3	OW 4
5	IB ST 24 PT 100 4/4	-
6	IBS IP DIO	OW 5
7	IBS IP CBK	-
8	IBS IP CDI	-
9	IBS IP CDO	OW 6
10	IBS IP CDI	-
11	IBS 24 BK I/O-T	OW 7
12	IBS 24 DI/32	-
13	IBS AI 3	OW 8, OW 9, OW 10, OW 11

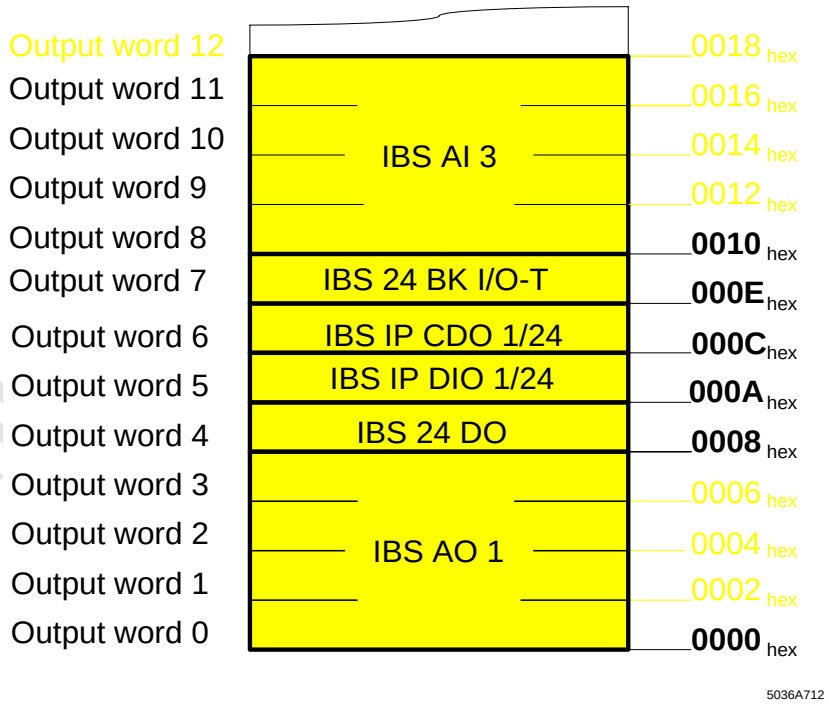


Figure 6-5: Output addresses in the memory map (OUT buffer)
The start addresses of the modules are in bold type.

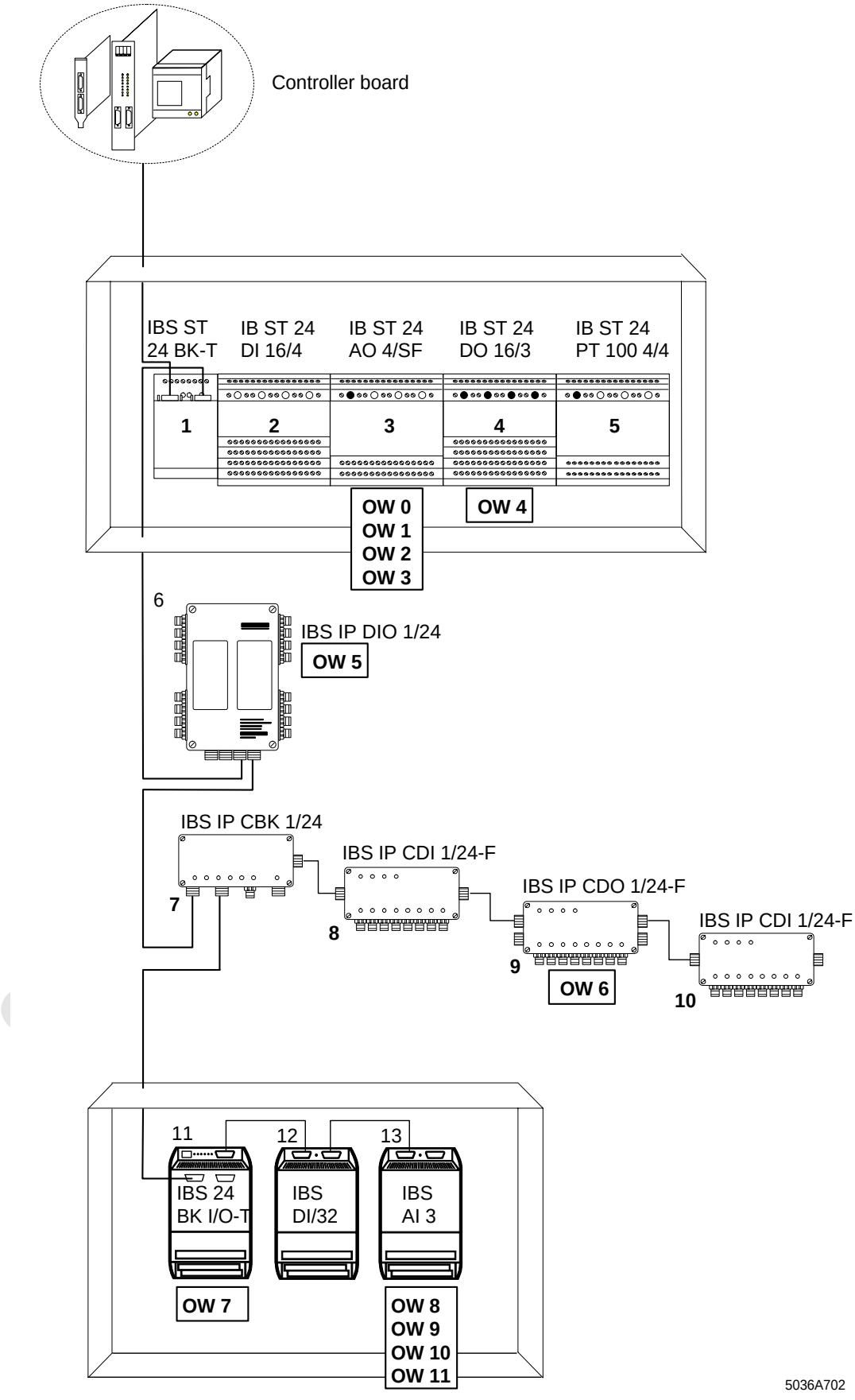


Figure 6-6: Output addresses under physical addressing

6.2.2 Command Sequence for Startup Under Physical Addressing

Table 6-4: Command sequence for startup under physical addressing

Action	Command	Code
Place outputs in safe state, stop data transmission on the bus	Alarmstop_Request	$004A_{hex}$
Clear register for diagnostic bits	Clear_Display_Request	$004E_{hex}$
Configure bus system	Configure_BUS_Request	0023_{hex}
Check configuration	Check_Physical_Configuration_Request	0058_{hex}
Start data cycles	Start_BUS_Cycle_Request	0001_{hex}

Section 8 describes the commands in detail.

6.3 Logical Addressing of IBS Devices

Unlike physical addressing, logical addressing allows to assign the addresses of the IBS devices independent of the actual physical order of their arrangement in the bus system. The desired addresses are assigned by assignment list entries in the order of the physical arrangement of IBS devices.

The following assigment lists are required for logical addressing:
- List with length and ID codes of all devices of your bus configuration
- List of the logical bus segments
- List of the logical input addresses
- List of the logical output addresses

The lists are transferred to the controller board, where they are checked and stored in the RAM. The controller board addresses the IBS devices according to this list during startup.

Therefore, logical addressing allows an arbitrary assignment of the I/O data from the InterBus-S system to the memory addresses in the multi-port memory and, therefore, in the host. The fact that the addresses can be freely selected makes it easier to enable or disable system parts in the bus configuration, as you need to change only the address list and not all addresses in your application program when adding or removing IBS devices.

In addition, further system-specific assignments are possible. For example, you can permanently assign the bus segment number to the system part (e.g. a switch cabinet number) or combine individual bus segments into groups.

Proceed as follows:
1. Clarify the following:
   - In what order are the IBS devices to be logically addressed?
   - Which IBS devices are to be assigned which addresses?
   - Which IBS devices are to be combined in a group address?
2. Create the logical IN address list in accordance with the preliminary considerations.
3. Create the logical OUT address list in accordance with the preliminary considerations.
4. If necessary, number the bus segments and carry out a group definition.
5. The controller board automatically checks the assignment lists for conformance and plausibility when receiving them (command *Implement_All_Logical_Address_Maps_Request* (0040_{hex})).

When the controller board has stored the assignment lists without errors, it can run InterBus-S with logical addressing.

6.3.1 Determining the Currently Connected Bus Configuration

The IBS command *Configure_Bus_Request* (0023_{hex}) causes the controller board to determine the currently connected bus configuration and to store it in the controller board RAM. The ID and length codes are read in for all connected InterBus-S devices.

The *Configure_Bus_Request* (0023_{hex}) command is described in detail in Section 8, *Commands for the IBS Master Board*.

All previously stored lists (addressing, group definitions, process data linkages and event definitions) will be deleted.

6.3.2 Checking the Bus Configuration

The IBS command *Check_Physical_Configuration_Request* (0058_{hex}) transfers a bus configuration required for the operation to the controller board in the form of length and ID codes. The controller board automatically compares the transferred configuration with the configuration stored in the RAM with the *Configure_BUS_Request* (0023_{hex}) command.

The *Check_Physical_Configuration_Request* command(0058_{hex}) is described in detail in Section 8, *Commands for the IBS Master Board*.

If the transferred configuration and the stored configuration are not identical, the controller board will generate an error message (CTRL ERR) indicating this condition. Using the *Send_Log_Address_Error_Request* ($005F_{hex}$) command you can then request a message on error type and error location.

Word 1	Code		
Word 2	Parameter count (n)		
Word 3	Length code	ID code	for device 1
Word 4	Length code	ID code	for device 2
Word 5	Length code	ID code	for device 3
	...	...	
Word n+2	Length code	ID code	for device n
Bit	15 8	7 0	

Figure 6-7: *Check_Physical_Configuration_Request* command format

Key:

Code:	Command code (here: 0058_{hex})
Parameter count:	Number of subsequent words (here: number of devices).
Length code:	The length code describes the address space requirements of the IBS device in the host.
ID code:	Identification code (short: ID code) of the IBS device

Use in addition the command *Receive_Local_Bus_Code_Map_Request* if your bus configuration contains installation remote bus devices.

Buffer in the programming language "C":

```
/*********************************************************************/
/* "Check_Physical_Configuration_Request"                            /
/*  as a list for the bus configuration                              /
/*********************************************************************/
USIGN16 chk_phy_cnf[] = //Data field of the unsigned-integer type
              {0x0058,  //Command Code
               0x000D,  //Parameter Count
               0x0008,  //IBS ST 24 BK-T
               0x01BE,  //IB ST 24 DI 16/4
               0x047D,  //IB ST 24 AO 4/SF
               0x01BD,  //IB ST 24 DO 16/3
               0x047B,  //IBS PT 100
               0x0107,  //IBS IP DIO 1/24
               0x000C,  //IBS IP CBK 1/24
               0x810A,  //IBS IP CDI 1/24
               0x8109,  //IBS IP CDO 1/24
               0x810A,  //IBS IP CDI 1/24
               0x010B,  //IBS 24 BK I/O-T
               0x028E,  //IBS 24 DI/32
               0x0447}; //IBS AI 3
```

Buffer in the programming language "Pascal":

```
{*********************************************************************}
{* "Check_Physical_Configuration_Request"                            *}
{*  as a list for the bus configuration                              *}
{*********************************************************************}

const chk_phy_cnf : array[1..15] of word =
                        {Data field of the unsigned-integer type}
              ($0058,   {Command Code}
               $000D,   {Parameter Count}
               $0008,   {IBS ST 24 BK-T}
               $01BE,   {IB ST 24 DI 16/4}
               $047D,   {IB ST 24 AO 4/SF}
               $01BD,   {IB ST 24 DO 16/3}
               $047B,   {IBS PT 100}
               $0107,   {IBS IP DIO 1/24}
               $000C,   {IBS IP CBK 1/24}
               $810A,   {IBS IP CDI 1/24}
               $8109,   {IBS IP CDO 1/24}
               $810A,   {IBS IP CDI 1/24}
               $010B,   {IBS 24 BK I/O-T}
               $028E,   {IBS 24 DI/32}
               $0447); {IBS AI 3}
```

6.3.3 Assignment of Logical Bus Segment Numbers

The IBS command *Receive_Local_Bus_Code_Map_Request* (0069_{hex}) carries out the arbitrary assignment of the bus segment number to the InterBus-S devices with bus terminal module functionality (remote bus and installation remote bus devices). This is useful to avoid the need to change the bus segment numbering in the current system.

Wort 1	Code			
Wort 2	Parameter count			
Wort 3	RB level	0_{hex}	Bus segment number	Device 1
Wort 4	RB level	0_{hex}	Bus segment number	Device 2
	...	...	...	
Wort n+2	RB level	0_{hex}	Bus segment number	Device n

Bit	15 12	11 8	7 4	3 0

Figure 6-8: *Receive_Local_Bus_Code_Map_Request* command format

Key:

Code: Command code (here: 0069_{hex})

Parameter count: Number of subsequent words (here: number of bus segments).

RB level: Enter the RB level here:
- Main line: RB level = 0_{hex} (0000_{bin})
- Remote bus branch (e.g. branching off from IBS IP CBK):
  RB level = 1_{hex} (0001_{bin})

Bus segment number: Enter here the desired logical bus segment numbers in the order of their physical arrangement.

The list becomes valid only after a successful execution of the *Implement_All_Logical_Address_Maps_Request* (0040_{hex}) program.

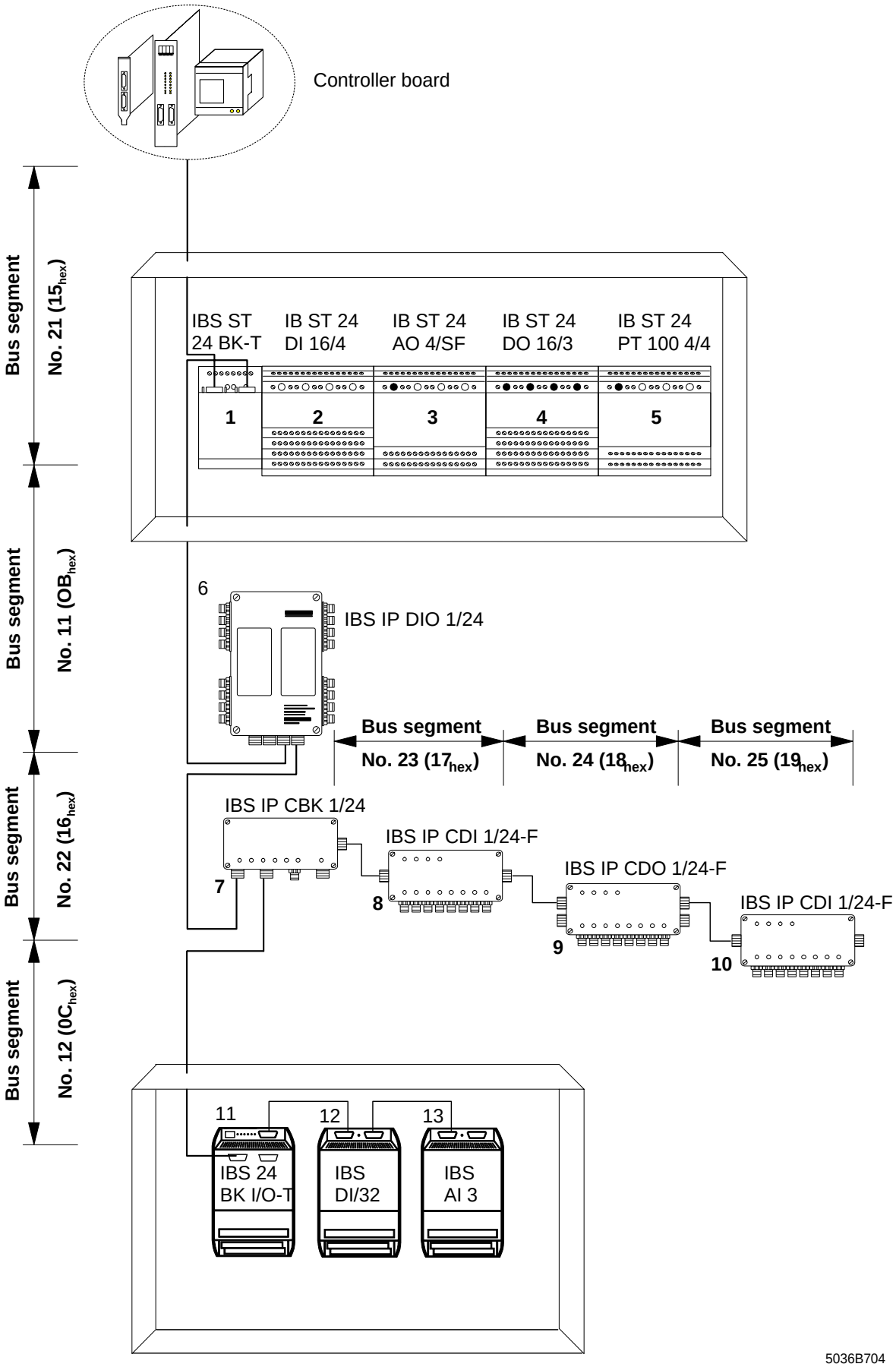


Figure 6-9: Numbering of the bus segments under logical addressing

Buffer in the programming language "C":

```
/*****************************************************************/
/* "Receive_Local_Bus_Code_Map_Request"                          */
/*  to implement logical bus segment numbers                     */
/*****************************************************************/
USIGN16 log_lb_adr_map[] =
              {0x0069,  //Command Code
               0x000D,  //Parameter Count
               0x0015,  //IBS ST 24 BK-T
               0x0000,  //IB ST 24 DI 16/4
               0x0000,  //IB ST 24 AO 4/SF
               0x0000,  //IB ST 24 DO 16/3
               0x0000,  //IBS PT 100
               0x000B,  //IBS IP DIO 1/24
               0x0016,  //IBS IP CBK 1/24
               0x1017,  //IBS IP CDI 1/24
               0x1018,  //IBS IP CDO 1/24
               0x1019,  //IBS IP CDI 1/24
               0x000C,  //IBS 24 BK I/O-T
               0x0000,  //IBS 24 DI/32
               0x0000}; //IBS AI 3
```

Buffer in the programming language "Pascal":

```
{*****************************************************************}
{* "Receive_Local_Bus_Code_Map_Request"                          *}
{*  to implement logical bus segment numbers                     *}
{*****************************************************************}
const log_lb_adr_map :array[1..15] of word =
              ($0069,  {Command Code}
               $000D,  {Parameter Count}
               $0015,  {IBS ST 24 BK-T}
               $0000,  {IB ST 24 DI 16/4}
               $0000,  {IB ST 24 AO 4/SF}
               $0000,  {IB ST 24 DO 16/3}
               $0000,  {IBS PT 100}
               $000B,  {IBS IP DIO 1/24}
               $0016,  {IBS IP CBK 1/24}
               $1017,  {IBS IP CDI 1/24}
               $1018,  {IBS IP CDO 1/24}
               $1019,  {IBS IP CDI 1/24}
               $000C,  {IBS 24 BK I/O-T}
               $0000,  {IBS 24 DI/32}
               $0000); {IBS AI 3}
```

6.3.4 Assignment of the Logical Addresses by the Programmer

Numbering of words when assigning logical address lists

InterBus-S numbers words in the counting mode which is common to **p**rogrammable **l**ogic **c**ontrollers (PLCs). As successive words start at even byte addresses, they are numbered on the basis of these even byte addresses. Thus, for example, the word consisting of bytes 6 and 7 is given the number 6.

Table 6-5: Differences in word numbering modes between programmable logic controllers (PLC) and PCs

PLC	
Word	**Byte**
Word n	Byte n+1
	Byte n
...	...
	...
Word 6	Byte 7
	Byte 6
Word 4	Byte 5
	Byte 4
Word 2	Byte 3
	Byte 2
Word 0	Byte 1
	Byte 0

PC	
Word	**Byte**
Word n/2	Byte n+1
	Byte n
...	...
	...
Word 3	Byte 7
	Byte 6
Word 2	Byte 5
	Byte 4
Word 1	Byte 3
	Byte 2
Word 0	Byte 1
	Byte 0

Therefore, use only even addresses when creating address lists with the commands
- *Receive_Logical_IN_Address_Map_Request* ($003A_{hex}$) and
- *Receive_Logical_OUT_Address_Map_Request* ($003B_{hex}$)

for the start addresses of IBS devices with an address area of 16 bits (or more).

IBS devices with an address area of 8 bits (length code 81_{hex}) can be placed at either even or odd start addresses. Thus, you can combine two 8-bit modules into one word in the logical addressing mode.

The typical InterBus counting mode for words (Table 6-5, left) only needs to be observed when creating the logical assignment lists (logical addressing, process data linkage, event definition). The driver software (functions *DDI_DTI_ReadData* and *DDI_DTI_WriteData*) employs the word counting mode that is usually used by high-level languages (Table 6-5, right) when placing the data in the buffer (see, for example, Figure 6-12).

6.3.4.1 Assignment of the Logical Input Addresses

The IBS command *Receive_Logical_IN_Address_Map_Request* ($003A_{hex}$) transfers the list of logical input addresses to the controller board.

Word 1	Code			
Word 2	Parameter count			
Word 3	DC	0000_{bin}	Input address	Device 1
Word 4	DC	0000_{bin}	Input address	Device 2
	...	0000_{bin}	...	
Word n+2	DC	0000_{bin}	Input address	Device n
Bit	15 12	11 9	8 0	

Figure 6-10: *Receive_Logical_IN_Address_Map_Request* command format

Key:

Code: Command code (here: $003A_{hex}$)

Parameter count: Number of subsequent words (here: number of devices).

Input address: Enter here (10 bits) the desired address (as byte address) for all modules with process input data. With modules with more than one process input word (e.g. with 32 bits), the next higher addresses are automatically also assigned.

DC: Enter here the data consistency for the process data access. The data consistency ensures that the specified data width is from one IBS cycle. The default value for I/O modules is 16 bits (00_{bin}). For IBS devices requiring the coherent transmission of greater data widths, the data consistency has to be increased. Such devices are, for example, encoders, operator interfaces or analog modules with a resolution of more than 16 bits. The following values are permissible for the data consistency:

- 00_{bin} 16 bits (standard)
- 01_{bin} 32 bits (e.g. encoders, operator interfaces)
- 10_{bin} 8 bits (only for modules with a length code of 81_{hex}, to which byte accesses are to be made!)
- 11_{bin} 48 bits (e.g. encoders, operator interfaces)

Enter 0000_{hex} for all modules without process input data (e.g. dedicated bus terminal modules or dedicated output modules).

Place the start addresses of IBS devices with address areas of 16 bits (or more) only at even addresses.

The list becomes valid only after a successful execution of the command *Implement_All_Logical_Address_Maps_Request* command (0040_{hex}).

Buffer in the programming language "C":

```
/*********************************************************************/
/* "Receive_Logical_IN_Address_Map_Request"                          */
/*  to implement all in addresses                                    */
/*********************************************************************/
USIGN16 in_adr_map[] =
              {0x003A,  //Command Code
               0x000D,  //Parameter Count
               0x0000,  //IBS ST 24 BK-T
               0x0000,  //IB ST 24 DI 16/4
               0x0000,  //IB ST 24 AO 4/SF
               0x0000,  //IB ST 24 DO 16/3
               0x001C,  //IBS PT 100
               0x000A,  //IP DIO 1/24
               0x0000,  //IP CBK 1/24
               0x0002,  //IP CDI 1/24
               0x0000,  //IP CDO 1/24
               0x0004,  //IP CDI 1/24
               0x000C,  //IBS 24 BK I/O-T
               0x0006,  //IBS 24 DI/32
               0x0014}; //IBS AI 3
```

Buffer in the programming language "Pascal":

```
{*********************************************************************}
{* "Receive_Logical_IN_Address_Map_Request"                          *}
{*  to implement all in addresses                                    *}
{*********************************************************************}
const in_adr_map : array [1..15] of word =
              ($003A,  {Command Code}
               $000D,  {Parameter Count}
               $0000,  {IBS ST 24 BK-T}
               $0000,  {IB ST 24 DI 16/4}
               $0000,  {IB ST 24 AO 4/SF}
               $0000,  {IB ST 24 DO 16/3}
               $001C,  {IBS PT 100}
               $000A,  {IP DIO 1/24}
               $0000,  {IP CBK 1/24}
               $0002,  {IP CDI 1/24}
               $0000,  {IP CDO 1/24}
               $0004,  {IP CDI 1/24}
               $000C,  {IBS 24 BK I/O-T}
               $0006,  {IBS 24 DI/32}
               $0014); {IBS AI 3}
```

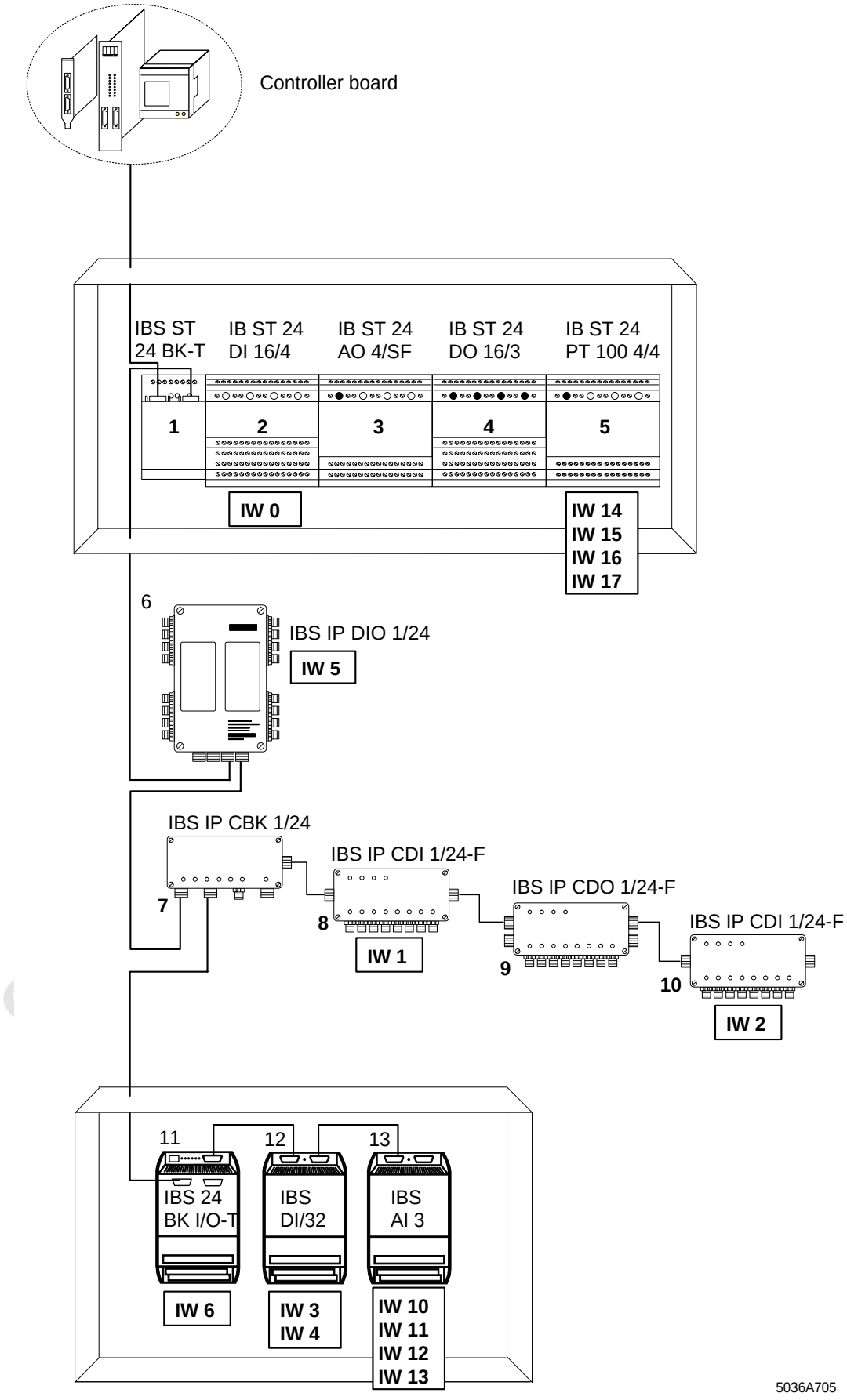


Figure 6-11: Input addresses under logical addressing

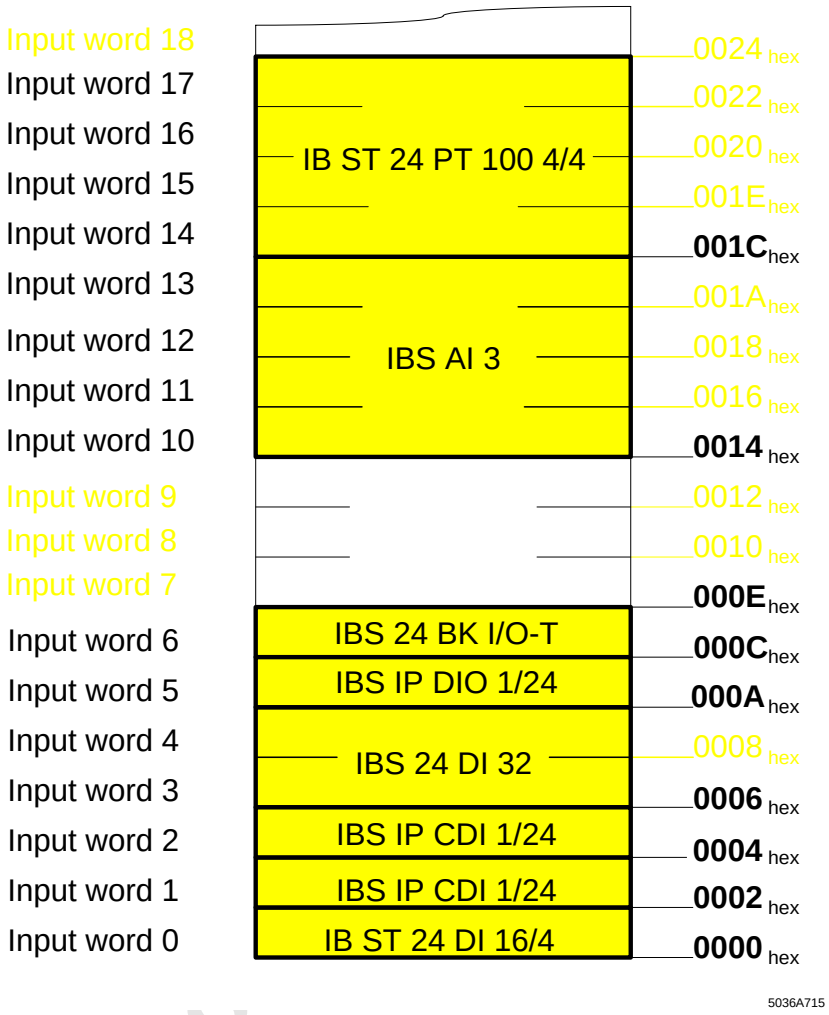


Figure 6-12: Input addresses in the memory map (IN buffer)
The start addresses of the modules are in bold type.

6.3.4.2 Assignment of the Logical Output Addresses

The IBS command *Receive_Logical_OUT_Address_Map_Request* ($003B_{hex}$) transfers the list of logical output addresses to the controller board.

Word 1	Code				
Word 2	Parameter count				
Word 3	0_{hex}	0 0 0	Output address		Device 1
Word 4	0_{hex}	0 0 0	Output address		Device 2
	...	...	...		
Word n+2	0_{hex}	0 0 0	Output address		Device n
Bit	15 12	11 9	8 0		

Figure 6-13: *Receive_Logical_OUT_Address_Map_Request* command format

Key:

Code: Command code (here: $003B_{hex}$)

Parameter count: Number of subsequent words (here: number of devices).

Output address: Enter here (10 bits) the desired address (as byte address) for all modules with process output data. With modules with more than one process output word (e.g. with 32 bits), the next higher addresses are automatically also assigned.

DC: Enter here the data consistency for the process data access. The data consistency ensures that the specified data width is from one IBS cycle. The default value for I/O modules is 16 bits (00_{bin}). For IBS devices requiring the coherent transmission of greater data widths, the data consistency has to be increased. Such devices are, for example, encoders, operator interfaces or analog modules with a resolution of more than 16 bits. The following values are permissible for the data consistency:

- 00_{bin} 16 bits (standard)
- 01_{bin} 32 bits (e.g. encoders, operator interfaces)
- 10_{bin} 8 bits (only for modules with a length code of 81_{hex}, to which byte accesses are to be made!)
- 11_{bin} 48 bits (e.g. encoders, operator interfaces)

Enter 0000_{hex} for all modules without process output data (e.g. dedicated bus terminal modules or dedicated input modules).

Place the start addresses of IBS devices with an address area of 16 bits (or more) only at even addresses.

The list will only be valid after a successful execution of the *Implement_All_Logical_Address_Maps_Request* (0040_{hex}) command.

Buffer in the programming language "C":

```
/*****************************************************************/
/* "Receive_Logical_OUT_Address_Map_Request"                     */
/*  to implement all out addresses                               */
/*****************************************************************/
USIGN16 out_adr_map[] =
             {0x003B,  //Command Code
              0x000D,  //Parameter Count
              0x0000,  //IBS ST 24 BK-T
              0x0000,  //IB ST 24 DI 16/4
              0x0014,  //IB ST 24 AO 4/SF
              0x0000,  //IB ST 24 DO 16/3
              0x0000,  //IBS PT 100
              0x0004,  //IBS IP DIO 1/24
              0x0000,  //IBS IP CBK 1/24
              0x0000,  //IBS IP CDI 1/24
              0x0002,  //IBS IP CDO 1/24
              0x0000,  //IBS IP CDI 1/24
              0x0006,  //IBS 24 BK I/O-T
              0x0000,  //IBS 24 DI/32
              0x001C}; //IBS AI 3
```

Buffer in the programming language "Pascal":

```
{*****************************************************************}
{* "Receive_Logical_OUT_Address_Map_Request"                     *}
{*  to implement all out addresses                               *}
{*****************************************************************}
const out_adr_map : array[1..15] of word =
             ($003B,  {Command Code}
              $000D,  {Parameter Count}
              $0000,  {IBS ST 24 BK-T}
              $0000,  {IB ST 24 DI 16/4}
              $0014,  {IB ST 24 AO 4/SF}
              $0000,  {IB ST 24 DO 16/3}
              $0000,  {IBS PT 100}
              $0004,  {IBS IP DIO 1/24}
              $0000,  {IBS IP CBK 1/24}
              $0000,  {IBS IP CDI 1/24}
              $0002,  {IBS IP CDO 1/24}
              $0000,  {IBS IP CDI 1/24}
              $0006,  {IBS 24 BK I/O-T}
              $0000,  {IBS 24 DI/32}
              $001C); {IBS AI 3}
```

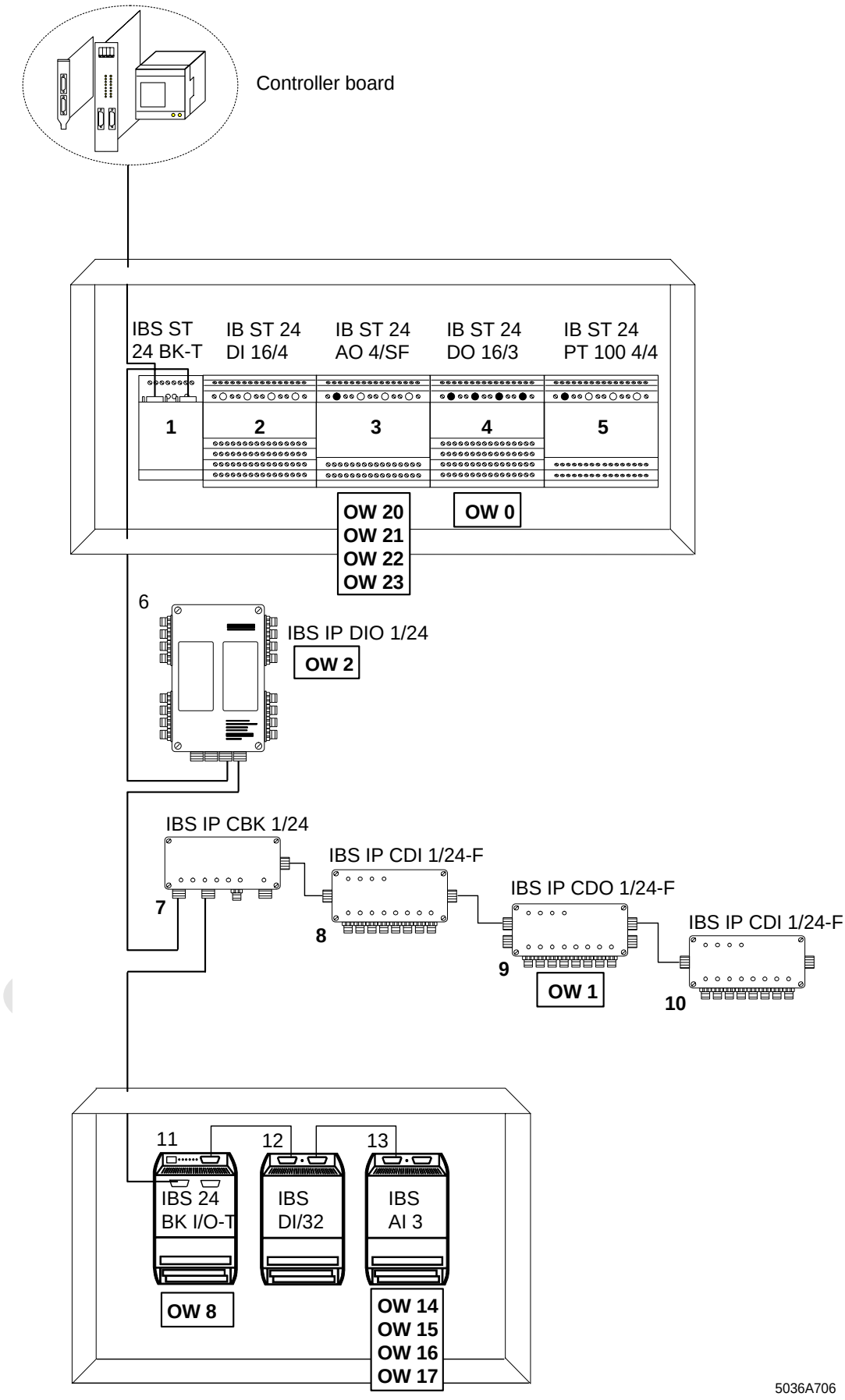


Figure 6-14: Output addresses under logical addressing

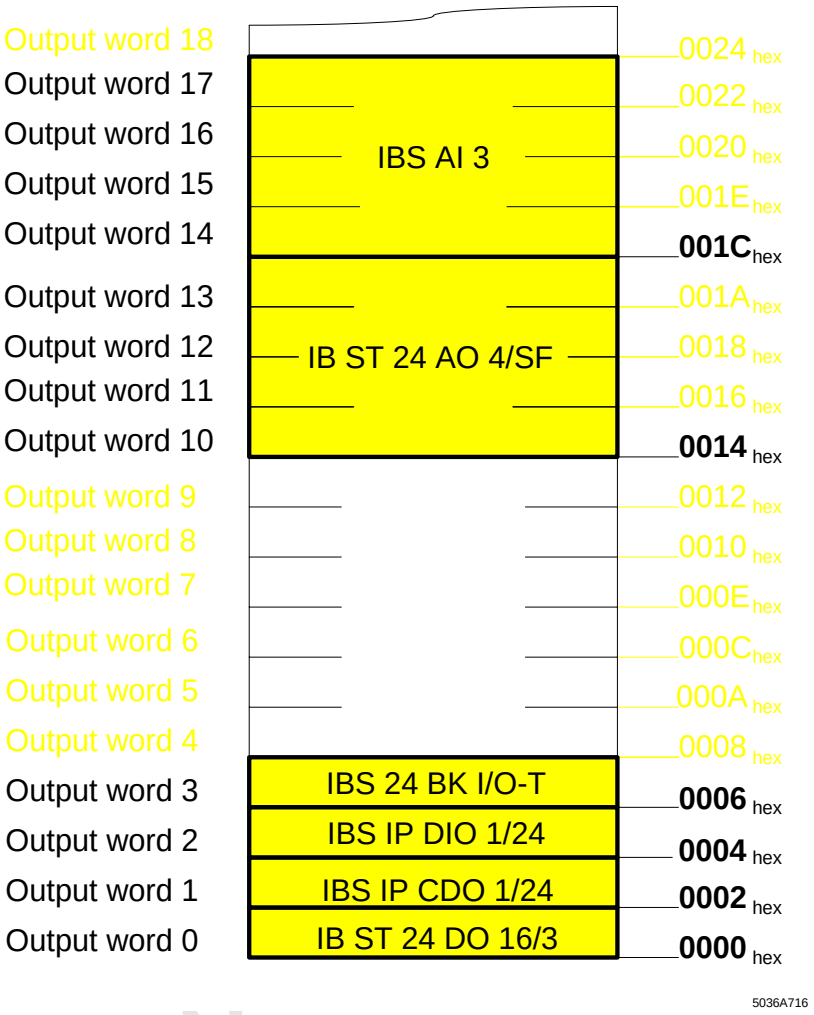


Figure 6-15: Output addresses in the memory map (OUT buffer)
The start addresses of the modules are in bold type.

6.3.4.3 Checking the Validity of the Assignment Lists

The *Implement_All_Logical_Address_Maps_Request* InterBus-S command (0040_{hex}) checks assignment lists transferred with IBS commands to the controller board for consistency and plausibility. If there are no errors in the lists, they will be accepted by the controller board and stored in the RAM.

It is recommended to call the *Implement_All_Logical_Address_Maps_Request* command (0040_{hex}) once more after every transfer of address lists (e.g. the commands 0069_{hex}, $003A_{hex}$, $003B_{hex}$) to be able to determine exactly in the event of an error which address list is faulty.

In the event of an error, lists transferred before will not be accepted. The negative acknowledgment of this command is *Logical_Address_Error* ($002B_{hex}$).

The *Send_Log_Address_Error_Request* command ($005F_{hex}$) creates a message with detailed information on the cause of the error.

The *Implement_All_Logical_Address_Maps_Request* (0040_{hex}) command may only be used when the bus is in the stop state. If necessary, call the *Alarm_Stop_Request* ($004A_{hex}$) command before calling this command.

6.3.5 Command Sequence for Startup Under Logical Addressing

Table 6-6: Command sequence for startup under logical addressing

Action	Command	Code
Place outputs in safe state, stop data transfer on the bus	Alarmstop_Request	$004A_{hex}$
Clear register for diagnostic bits	Clear_Display_Request	$004E_{hex}$
Configure bus system	Configure_BUS_Request	0023_{hex}
Check configuration	Check_Physical_Configuration_Request	0058_{hex}
Number bus segments	Receive_Logical_Local_Bus_Address_Map_Request	0069_{hex}
Check validity of assignment lists	Implement_All_Logical_Address_Maps_Request	0040_{hex}
Transfer logical IN address list	Receive_Logical_IN_Address_Map_Request	$003A_{hex}$
Check validity of assignment lists	Implement_All_Logical_Address_Maps_Request	0040_{hex}
Transfer logical OUT address list	Receive_Logical_OUT_Address_Map_Request	$003B_{hex}$
Check validity of assignment lists	Implement_All_Logical_Address_Maps_Request	0040_{hex}
Start of data cycles	Start_BUS_Cycle_Request	0001_{hex}

Section 8 describes the commands in detail.

It is recommended to call the *Implement_All_Logical_Address_Maps_Request* (0040_{hex}) once more after every transfer of address lists (e.g. the commands 0069_{hex}, $003A_{hex}$, $003B_{hex}$) to be able to determine exactly in the event of an error which address list is faulty.

6.4 Group Definition

6.4.1 Creating Functional Groups

When, for example, certain system parts are to be disabled while other system parts are to keep running, combine bus segments into functional groups before. In the following, individual groups can be disabled or enabled by executing the InterBus-S commands *Switch_Group_Off_Request* (0021_{hex}) and *Switch_Group_On_Request* (0020_{hex}). The IBS command *Receive_Group_Numbers_Request* (0049_{hex}) creates groups. Assign a group number of between 0 and 255 to each bus segment.

Word 1	Code		
Word 2	Parameter count		
Word 3	00_{hex}	Bus segment number	for 1st bus segm.
Word 4	00_{hex}	Group	for 1st bus segm.
Word 5	00_{hex}	Bus segment number	for 2nd bus segm.
Word 6	00_{hex}	Group	for 2nd bus segm.
	...	...	
	...	...	
Word n+1	00_{hex}	Bus segment number	for xth bus segm.
Word n+2	00_{hex}	Group	for xth bus segm.
Bit	15 8	7 0	

Figure 6-16: *Receive_Group_Numbers_Request* command format

Key:

Code:	Command code (here: 0049_{hex})
Parameter count:	Number of subsequent words (here: number of bus segments multiplied by 2)
Bus segment number:	Number of the bus segment that is assigned to the group specified in the next line
Group:	Number of the group to which the bus segment specified in the previous line is assigned (0 to 255_{dec} corresponding to 00_{hex} to FF_{hex})

Enter only IBS devices which are on the main line of the bus configuration (RB level 0). Devices in the remote bus branch line (RB level 1) are included in the group definition by assigning the bus terminal module opening this branch line to a group.

This example uses the bus segment numbers assigned in Section 6.3.3 (see Figure 6-9).

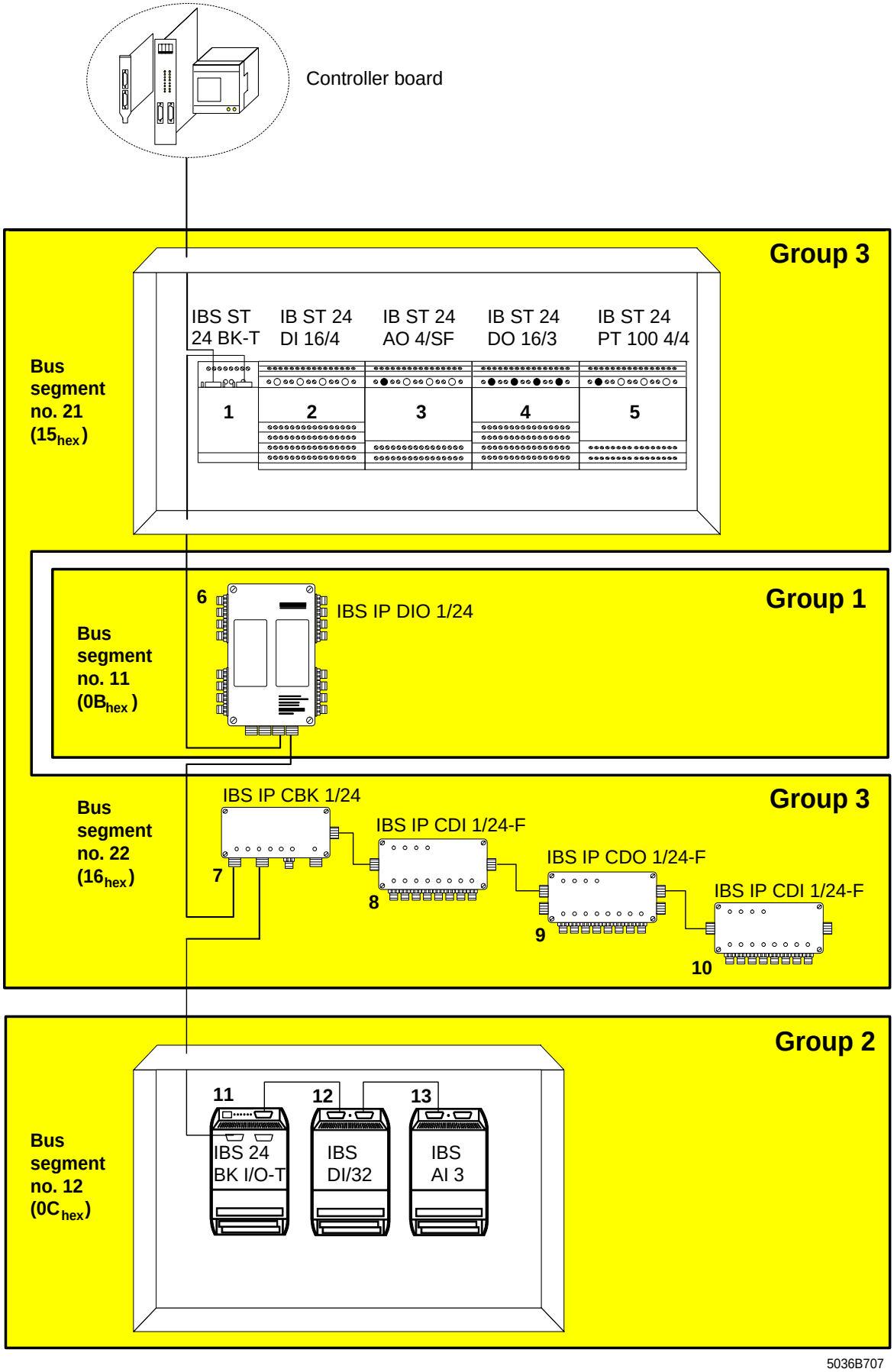


Figure 6-17: Division into logical functional groups

Buffer in the programming language "C":

```
/*****************************************************************/
/* "Receive_Group_Numbers_Request" to implement logical groups   */
/*****************************************************************/
USIGN16 grp_num[] =
             {0x0049, // Command Code
              0x0008, // Parameter Count
              0x0015, // lb number
              0x0003, // group number to which this lb should belong
              0x000B, // lb number
              0x0001, // group number to which this lb should belong
              0x0016, // lb number
              0x0003, // group number to which this lb should belong
              0x000C, // lb number
              0x0002};// group number to which this lb should belong
```

Buffer in the programming language "Pascal":

```
{*************************************************************}
{* "Receive_Group_Numbers_Request" to implement logical groups *}
{*************************************************************}
const grp_num : array[1..10] of word =
             ($0049, {Command Code}
              $0008, {Parameter Count}
              $0015, {lb number}
              $0003, {group number to which this lb should belong}
              $000B, {lb number}
              $0001, {group number to which this lb should belong}
              $0016, {lb number}
              $0003, {group number to which this lb should belong}
              $000C, {lb number}
              $0002);{group number to which this lb should belong}
```

6.4.2 Swtiching Groups Off

The command *Switch_Group_Off_Request* (0021_{hex}) is used to disable (switch off) a particular group. The outputs of the IBS devices in the disabled bus segment and the associated inputs of the PLC or host computer are set to 0.

Word1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Group number

Bit	15 8	7 0

Figure 6-18: *Switch_Group_Off_Request* format

Key:

Code:	Command code (here: 0021_{hex})
Parameter count:	Number of subsequent words (here: 1)
Group number:	Number of the group to be disabled. (00_{hex} to FF_{hex} corresponds to 0 to 255_{dec})

When a system part is disabled, the supply voltage for the module electronics (logic voltage) must be retained for the IBS devices with bus terminal module functionality included in this system part, to ensure that the operation of the remaining bus configuration can continue.

Buffer in the programming language "C":

```
/*****************************************************************/
/* "Switch_Group_Off_Request" to switch off a group              */
/*****************************************************************/
USIGN16 grp_off[] =
             {0x0021,  // Command Code
              0x0001,  // Parameter Count
              0x0003}; // switch off group number 1
```

Buffer in the programming language "Pascal":

```
{*****************************************************************}
{* "Switch_Group_Off_Request" to switch off a group              *}
{*****************************************************************}
const grp_off : array[1..3] of word =
             ($0021,  {Command Code}
              $0001,  {Parameter Count}
              $0003); {switch off group number 1}
```

6.4.3 Enabling Groups On

The *Switch_Group_On_Request* (0020_{hex}) command is used to re-enable (switch on) a group which had been disabled (switched off) before.

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Group number

Bit	15 8	7 0

Figure 6-19: the *Switch_Group_On_Request* command format

Key:

Code:	Command code (here: 0020_{hex})
Parameter count:	Number of subsequent words (here: 1)
Group number:	Number of the group to be enabled (00_{hex} to FF_{hex} corresponds to 0 to 255_{dec})

Buffer in the programming language "C":

```
/*********************************************************************/
/* "Switch_Group_On_Request" to switch on a group                    */
/*********************************************************************/
USIGN16 grp_on[] =
             {0x0020,  // Command Code
              0x0001,  // Parameter Count
              0x0003}; // switch on group number 1
```

Buffer in the programming language "Pascal":

```
{*********************************************************************}
{* "Switch_Group_On_Request" to switch on a group                    *}
{*********************************************************************}
const grp_on : array[1..3] of word =
             ($0020,  {Command Code}
              $0001,  {Parameter Count}
              $0003); {switch on group number 1}
```

6.4.4 Defining the Handling of Groups in the Event of Errors

After an error in a group (bus STOP), the *Define_Group_Error_Characteristics_Request* (0060_{hex}) command defines for each group whether the bus is to restart automatically without this group.

Word 1	Code			
Word 2	Parameter count			
Word 3	Mode	0_{hex}	Group number	for group 1
Word 4	Mode	0_{hex}	Group number	for group 2
	...	...	...	
Word n+2	Mode	0_{hex}	Group number	for group n
Bit	15 12	11 8	7 0	

Figure 6-20: *Define_Group_Error_Characteristics_Request* command format

Key:

Code: Command code

Parameter count: Number of subsequent words

Mode: Handling of the remaining parts of the bus system in the event of an error in the group described by this line
0_{hex} The complete bus remains at STOP in the event of an error.
8_{hex} The bus is restarted without the faulty group.

Group number: Group number of the group described by this line
(00_{hex} to FF_{hex} corresponds to 0 to 255_{dec})

Buffer in the programming language "C":

```
/*****************************************************************/
/* "Define_Groups_Error_Characteristics_Request"                 */
/*  for restart after an error                                   */
/*****************************************************************/
USIGN16 grp_err_chr[] =
            {0x0060,  // Command Code
             0x0003,  // Parameter Count
             0x8001,  // Group 1
             0x8002,  // Group 2
             0x8003}; // Group 3
```

Buffer in the programming language "Pascal":

```
{*****************************************************************}
{* "Define_Groups_Error_Characteristics_Request"                 *}
{*  for restart after an error                                   *}
{*****************************************************************}
const grp_err_chr : array[1..5] of word =
             ($0060,  {Command Code}
              $0003,  {Parameter Count}
              $8001,  {Group 1}
              $8002,  {Group 2}
              $8003); {Group 3}
```

Section 7

Error Diagnostics

This section provides information on the diagnostic analysis and correction

- of errors in the InterBus-S system (controller board, bus configuration);
- of errors in the application program.

7 Error Diagnostics

InterBus-S provides comprehensive error diagnostics.

7.1 Hardware Diagnostics

7.1.1 Diagnostic Indicators on the Controller Board

The controller board has 4 LEDs for easy status diagnostics on its PC board holder.

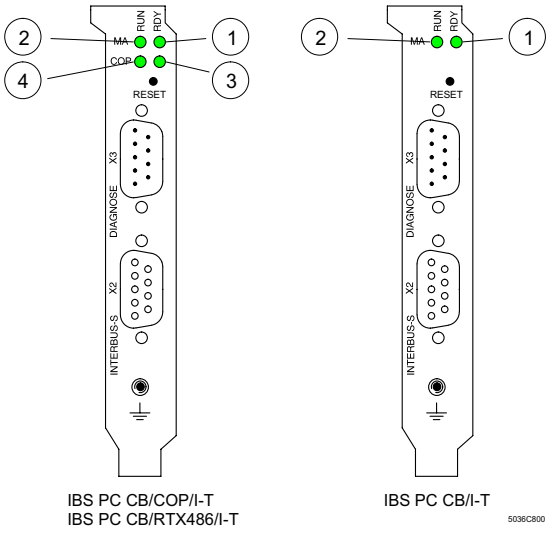


Figure 7-1: Diagnostic Indicators on the PC board holder

1 External *MA READY* LED (green)
2 External *MA RUN* LED (green)
3 External *COP READY* LED (green)
4 External *COP RUN* LED (green)

IBS master board

MA READY: Following power-on, the IBS master board has executed a boot check for all functional units including MPM without detecting errors, and is ready for operation.

MA RUN: The IBS master board has started InterBus-S. ID or data cycles are being transmitted.

Coprocessor board
(not IBS PC CB/I-T)

COP RUN: The coprocessor board operating system has booted; an application program can be started.

COP READY: A program is running on the coprocessor board.

Coprocessor board booting is followed by the automatic start of various utilities, some of which stay resident as TSR programs in the COP memory. Therefore, the *COP READY* LED is lit after the complete system has come up.

7.1.2 Diagnostic Indicators on Bus Terminal Modules

The bus terminal modules, too, also provide local status diagnostics by means of LEDs. Figure 8-2 shows the diagnostics on the bus terminal module IBS 24 BK-T as an example.

Typical indicators on BK modules:
- Supply voltage
- Remote bus monitoring
- Local bus monitoring
- InterBus-S status indicator

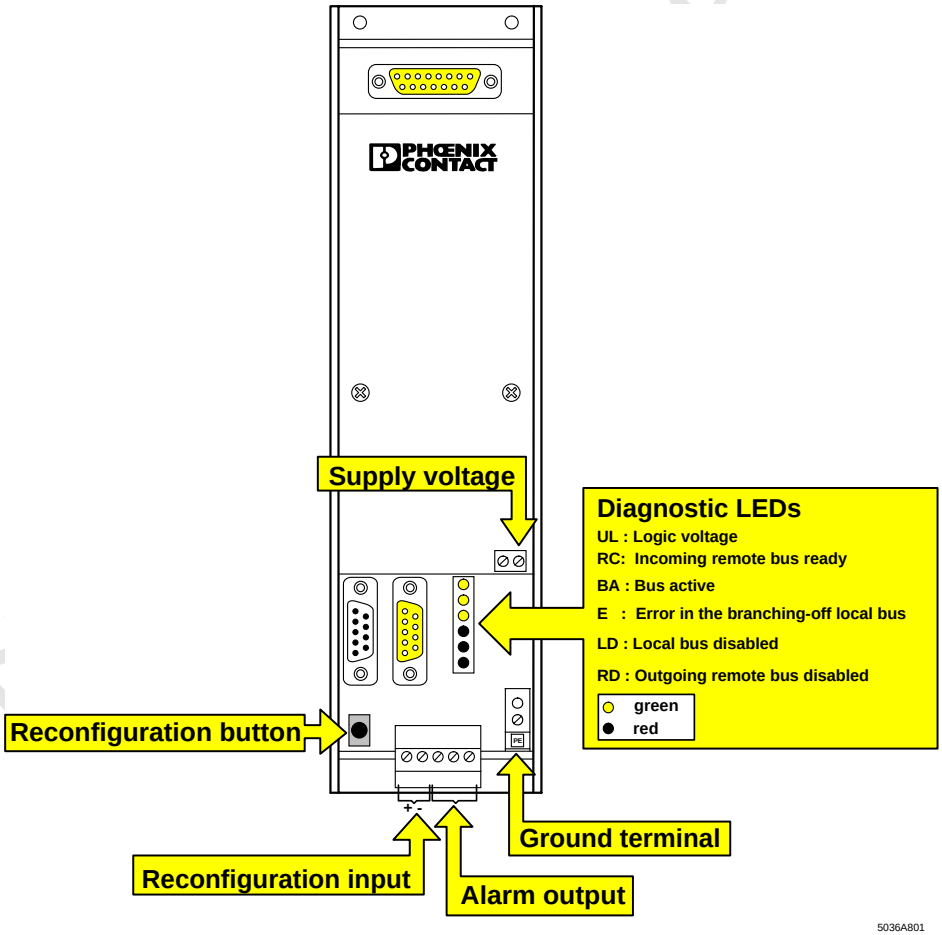


Figure 7-2: LED diagnostics on bus terminal modules

7.1.3 Diagnostic Indicators on IBS Devices with I/O Functions

The IBS devices with I/O function also feature status and diagnostic LEDs to allow quick local diagnostics. The diagnostic LEDs provide information on the type and the location of the error that has occurred. The status LEDs indicate the input and output (I/O) states.

Typical indicators on IBS devices with I/O function:

- Supply voltage
- Output overload/short circuit (for each group, individual DI/DOs)
- Status LED for each I/O channel

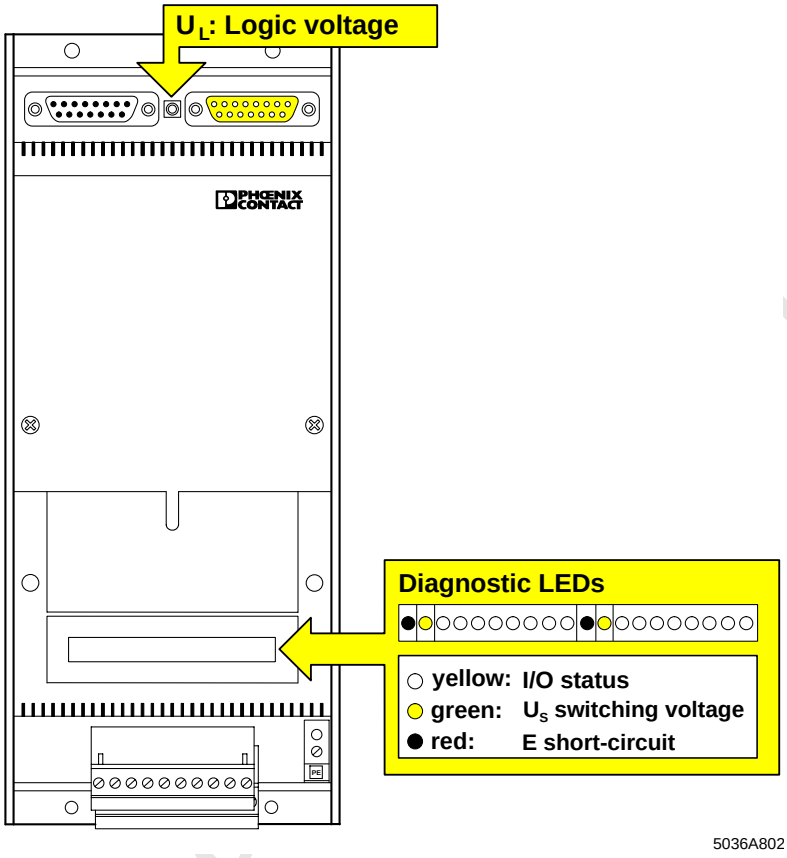


Figure 7-3: LED diagnostics on I/O modules

7.1.4 Diagnostics of IBS Devices from Other Manufacturers

The diagnostic indicators on other manufacturers' IBS devices are divided into two groups of indicators:

- Bus-specific indicators:
  These indicators correspond mainly to those on the bus terminal modules or IBS devices with I/O function from Phoenix Contact
- Device-specific indicators:
  These indicators depend on the IBS device. Please consult your device manual or contact the manufacturer of the IBS device.

7.2 Diagnostics with Software Tools

Various diagnostic software tools are available, which allow to access your bus configuration without major programming efforts.

- The process data monitor program *PCCBMONI.EXE* (is supplied with the driver software)
- The diagnostic and configuration software *IBS SYS SWT* under DOS (Order No. 27 52 73 8)
- The InterBus manager *IBS CMD SWT*, a graphics-oriented program, which allows to configure, monitor and diagnose your IBS system under Microsoft Windows® (Order No. 27 50 97 6).

7.2.1 The Process Data Monitor Program

A process data monitor program is supplied with the driver software for IBS PC CB/.../I-T.

Load the *PCCBMONI.EXE* file from the driver diskette into a directory on your hard disk, from where you start the process data monitor program by calling *PCCBMONI*. The program provides the following services:

- First startup of an InterBus-S system without any programming
- Test of your bus configuration
- Check of the connected configuration
- Indication of the status of individual inputs (binary)
- Indication fo the status of input words (hexadecimal)
- Setting of individual outputs (binary)
- Setting of output words (hexadecimal)
- Resetting of the InterBus-S system

The operation of the monitor program is described in detail in Section 4.

7.2.2 The Diagnostic and Configuration Software IBS SYS SWT

The*IBS SYS SWT* program (Order No. 27 52 73 8) is a user interface for InterBus-S on IBM-compatible PCs under DOS. It transfers parameterization data to the controller board and stores it there and allows easy, menu-driven parameterization and control of the controller board. With *IBS SYS SWT*, the functions of the IBS components (controller boards, modules, etc.) can be used without the need for programming.

The PC is coupled with *IBS SYS SWT* and the controller board via the serial interfaces (RS-232) of the two devices.

The individual program functions are:

- Transmission of commands to the controller board.
- Reception of messages from the controller board.
- Logging of the received messages and the transmitted commands.
- Storage of command sequences in a file, and transmission of these commands to the controller board.

- Creation and transfer of logical address lists to the controller board. These address lists cannot be permanently stored on the controller board and are lost when a controller board reset is carried out.
- Diagnostics and evaluation of the bus transmission.

IBS SYS SWT offers the user a simple interface to the controller board under DOS. Refer to the IBS SYS SWT UM E manual (Order No. 27 53 87 6) for further information.

7.2.3 The InterBus Manager IBS CMD SWT

The InterBus Manager *IBS CMD SWT* is a graphics-oriented program under Microsoft Windows®. It provides functions for InterBus configuration, startup and diagnostics. Dialog functions allow to operate and display (monitor) all connected IBS devices. In addition, the open structure of the user interface allows the integration of manufacturer- or device-specific operating and parameterization functions.

The individual program functions are:

- Creation of a bus configuration outline for setting up the address assignment between the host and the connected IBS devices
- Parameterization of complex IBS devices
- Dialog functions for data output (e.g. setting of outputs) and reading as well as displaying current input data (monitoring of inputs)
- Diagnostic functions for detecting and locating defective system parts (IBS devices, line runs, power etc.)
- Documentation of your IBS system by creating a system description including the parameterized device settings

Additional functions are provided by utilities from other IBS device manufacturers or IBS user groups (e.g. DRIVECOM). The user can easily integrate these utilities into the *IBS CMD SWT* user interface.

Therefore, the operating software *IBS CMD SWT* is an open platform for all IBS device manufacturers, which is a clearly arranged tool for configuration, monitoring and diagnostics of your InterBus-S system under Microsoft Windows® .

Refer to your manual IBS CMD SWT UM E (Order No. 27 53 95 7) for further information on *IBS CMD SWT*.

7.3 Diagnostics by the Application Program

7.3.1 Diagnostics of Controller Board and Bus Configuration

After the start of the IBS system by means of the *Start_Bus_Cycle_Request* command (0001_{hex}), the IBS master board operates the IBS system on its own and cyclically updates the I/O data in the controller board MPM. When an error occurs in the IBS system, the IBS master board sends a message. Depending on the type and the seriousness of the error, two different reactions are possible:

Minor errors which do not interfere with the operation of the IBS system, such as:

- Failure of the I/O supply voltage on an IBS device with error indication
- Overload of an output on an IBS device with error indication

In this case the IBS master board responds with, for example, the *Module-Error-Indication* ($80A0_{hex}$) message. Using the diagnostic function *GetIBSDiagnostic()*, the bus segment containing the defective IBS device can be determined. Using the command *Send_Localbus_Module_Error_Request* ($005B_{hex}$) you can determine the faulty IBS device in the bus segment to be entered as a parameter. The IBS system keeps operating.

Serious errors which do not allow any further operation of the IBS system, such as:

- Changes to the bus configuration, e.g. by removing a bus connector
- Bus cable defect (e.g. broken cable)
- Failure of the logic supply voltage (not I/O supply voltage) of a bus terminal module or of an IBS device with bus terminal functionality

In this case the IBS master board immediately stops the cyclic operation of the IBS system, resets all outputs and sends the message *Bus_System_Error_Indication* (0038_{hex}). Then the IBS master board searches through the IBS system and indicates the error causes in the message *Bus_Error_Information_Indication* ($80C4_{hex}$). Refer to Section 9 for further information on this message.

The message *Bus_Error_Information_Indication* (80C4) gives a very detailed description of the errors. An easier method is the use of the diagnostic function *GetIBSDiagnostic()*. It provides only the essential information and facilitates a quick diagnostic analysis of the IBS system. After the cause of the error has been removed (for example, the defective bus segment is disabled by means of a command), the IBS system can be restarted. (See Section 6 *Command Sequence for InterBus Startup*).

In addition, the diagnostic function can be used

- to determine, by inquiring the *Ready bit,* whether the IBS master board is ready again during a restart or after a warmstart with the *Warmstart_Request* (004C) command
- to determine, by inquiring the *Run bit,* whether the IBS master board is operating the bus and whether the I/O data is cyclically updated.

Further information on the messages

- Module_Error_Ind ($80A0_{hex}$),
- Bus_System_Error_Indication (0038_{hex}),
- Bus_Error_Information_Indication ($80C4_{hex}$)

and on further error messages is available in Section 9 *Messages of the IBS Master Board*.

7.3.1.1 Error Type

The controller board indicates the occurrence of an error on InterBus-S with the message *Bus_System_Error_Indication* (0038_{hex}). This message is followed by the message *Bus_Error_Information_Indication* ($80C4_{hex}$), which contains detailed information on the error condition of the bus system. The first parameter of this message specifies the error type. Firmware version 3.72 distinguishes six different error types.

Table 7-1: Error types in firmware version 3.72

Error type	Meaning	Cause	Remedy
EE01	Following the occurrence of the error, no error was found in the acquisition and comparison of the configuration.	- Cabling error - Shielding error	- Check the remote bus and local bus cabling. - Check the voltage supply. Tool: IBS SYS SWT or IBS CMD SWT
EE02	The maximum configuration was exceeded	Too many devices or register locations.	Check the addressing lists and the configuration. Up to 256 bus terminal modules and 256 words (total of all register length and of the PCP words) permissible on the data ring.
EE03	Error in a bus segment	- Defect in the bus terminal module or in the adjacent remote bus cable - Defective local bus cable or module	Correct the error causes. With the error EE03, the *diagnostic parameter register* that can be read out with the diagnostic function *GetIBSDiagnostic* indicates the number of the affected bus segment register instead of the error type)
EE04	Configuration acquisition failure.	IBS device does not respond.	Check the voltage supply for dips.
EE05	All groups were disabled.	Error in the application program.	Check the application program.
EE06	No data cycle possible, although no error occurred in the acquisition and comparison of the configuration	Module error	Locate the error with IBS SYS SWT, inform the Phoenix Contact Technical Support.

The register for diagnostic parameters, too, indicates the error types EE01, EE02, EE04, EE05 and EE06 in the event of an error. Instead of an error type, the number of the affected bus segment is indicated for the error EE03.

The description of the *Bus_Error_Information_Indication* message (80C4) in Section 9 contains further information on the six error types and their meanings, causes and remedies.

7.3.1.2 Meanings of Controller Board Error Numbers

Faulty operation of the controller board causes a controller board error number to be generated. The *GetIBSDiagnostic* diagnostic function (see the driver software manual) lets you read out this controller error number. The error numbers have the following meanings:

00_{hex}, 05_{hex}, 06_{hex}

Meaning: A firmware error was found on the controller board.

Remedy: Please consult Phoenix Contact.

07_{hex}

Meaning: An invalid command or PCP command was sent.

Remedy: Check the application program and remove undefined commands.

08_{hex}

Meaning: A command is to be followed by parameters, but another command follows.

Remedy: Check the command sequences in the application program. If necessary, add parameters or correct the command calls.

09_{hex}

Meaning: The number of parameters in the assignment lists is zero.

Remedy: Check the number of parameters in the definition lists.

$0A_{hex}$

Meaning: A non-defined command was sent.

Remedy: Check the application program and remove undefined commands.

Please consult Phoenix Contact should the error occur repeatedly.

$0B_{hex}$

Meaning: A firmware error was found on the controller board.

Remedy: Please consult Phoenix Contact.

$0C_{hex}$

Meaning: A message or the acknowledgment of a command is pending on the controller board and is not fetched in time.

Remedy: Call the function *DDI_MXI_RcvMessage* cyclically in the application program.

Comment: If you do not wish messages, you can issue the command *Disable_all_Messages_Request* (0048_{hex}).

$0D_{hex}$

Meaning: The controller board expects parameters which are not received during a given monitoring time.

Cause: The handshake did not take place in time.

Remedy: Initiate a host computer reset.

23_{hex}

Meaning: A bus error caused the watchdog to trip.

Remedy: Carry out a host computer reset.

Please consult Phoenix Contact should the error occur repeatedly.

26_{hex}, 27_{hex}, 28_{hex}, 29_{hex}

Meaning: A hardware error was found on the controller board.

Remedy: Please consult Phoenix Contact.

$2A_{hex}$

Meaning: A firmware error was found on the controller board.

Remedy: Please consult Phoenix Contact.

$2B_{hex}$

Meaning: An error was found in the address lists.

Consequences: The bus goes into the stop state without reset, the outputs are **not** reset.

Remedy: Check the assignment lists in the application program.

Meaning: An attempt was made to send address lists to the controller board while the bus was running.

Remedy: Switch the bus into the stop state before sending address lists to the controller board.

38_{hex}

Meaning: This error occurs in connection with a local bus error or a remote bus error. A test routine determines the defective segment.

Remedy: Check the bus configuration. Note the following diagnostic indications.

$3B_{hex}$

Meaning: End of cycle due to external signal.

Cause: The bus configuration is affected by interference.

Remedy: Check the bus configuration for interference and remove it. If the situation does not improve, please consult Phoenix Contact.

$3C_{hex}$

Meaning: A FiFo overflow was found.

Remedy: Carry out a host reset.

$3D_{hex}$

Meaning: A hardware error was found on the controller board.

Remedy: Carry out a host reset.

$3E_{hex}$, $3F_{hex}$, 40_{hex}, 41_{hex}, 42_{hex}, 43_{hex}, 44_{hex}, 45_{hex}

Meaning: A firmware error was found on the controller board.

Remedy: Please consult Phoenix Contact.

$4A_{hex}$

Meaning: The configuration stored in the controller board RAM (ID list) and the currently connected configuration are not identical.

Ursache: ID list is faulty or does not exist.

Remedy: Check the ID list.

Ursache: InterBus-S configuration does not exist.

Remedy: Connect the remote bus cable to the controller board.

$4B_{hex}$

Meaning: Too many or not enough parameters in the command or PCP command.

Remedy: Check the number of parameters in the application program.

$4C_{hex}$

Meaning: An error was found in the event definition.

Remedy: Check the event definition in the application program.

$4D_{hex}$

Meaning: A hardware error was found on the controller board.

Remedy: Carry out a controller board reset.

$4E_{hex}$

Meaning: You sent too many commands within a short time period.

Remedy: Reduce the number of issued commands per time period.

50_{hex}, 51_{hex}

Meaning: A firmware error was found on the controller board.

Remedy: Please consult Phoenix Contact.

55_{hex}

Meaning: An error was found in the group definition.

Remedy: Check the group definition in the application program.

57_{hex}, 58_{hex}

Meaning: A firmware error was found on the controller board.

Remedy: Please consult Phoenix Contact.

59_{hex}

Meaning: An undefined group number was specified in a command.

Remedy: Use only defined group numbers. Check the group definitions in the application program.

Meaning: An attempt was made to use a group disabling command for a group that could not be disabled.

Remedy: Use the group disabling commands only for groups that can really be disabled.

$5A_{hex}$

Meaning: A wrong bus segment number was specified when a bus terminal module alarm was enabled or disabled.

Remedy: Check the application program. Use only bus segment numbers that are defined and that really exist.

$5B_{hex}$

Meaning: An invalid length code is used in the application program.

Remedy: Check the length codes in the assignment lists.

65_{hex}

Meaning: The maximum number of communication modules in the bus has been exceeded.

Remedy: Reduce the number of communication modules to a maximum of 62.

66hex

Meaning: Invalid communication reference or wrong number of parameters

Remedy: Check the communication reference list (CR list):
- The *Parameter count* parameter must be identical with the number of devices.
- The CR list must have an ascending order and must have no gaps.

68_{hex}

Meaning: The ID list and the currently connected configuration are not identical.

Cause: If the error occurs during startup, it is probably due to an ID list error.

Remedy: Check the ID list.

Cause: If the error occurs during operation, the bus configuration is probably no longer complete or has been changed.

Remedy: Please check your bus configuration.

69_{hex}

Meaning: A PCP command was issued without prior initialization of the communication.

Remedy: Check the initialization of the communication in the application program.

$6B_{hex}$

Meaning: An error was found on the controller board. Please consult Phoenix Contact.

Section 8

Commands for the IBS Master Board

This section provides information on

- the tasks and the call methods of commands for the IBS master board;
- the parameters of the commands;
- the meanings and causes of messages in response to these commands.

8 Commands for the IBS Master Board

Table 8-1: Commands for the IBS master board

Code	IBS command	Page
0001_{hex}	Start_Bus_Cycle_Request	8-16
0002_{hex}	Stop_Bus_Cycle_Request	8-17
0006_{hex}	Send_Bus_Cycle_Counter_Request	8-27
0008_{hex}	Send_Software_Revision_Request	8-18
0020_{hex}	Switch_Group_On_Request	8-11
0021_{hex}	Switch_Group_Off_Request	8-10
0023_{hex}	Configure_Bus_Request	8-6
0024_{hex}	Set_BK_Alarm_Logical_Request	8-23
0025_{hex}	Reset_BK_Alarm_Logical_Request	8-24
0026_{hex}	Set_BK_Alarm_Physical_Request	8-24
0027_{hex}	Reset_BK_Alarm_Physical_Request	8-25
$002A_{hex}$	Receive_Events_Request	8-38
$002B_{hex}$	Enable_Event_Number_Request	8-40
$002C_{hex}$	Disableable_Event_Number_Request	8-41
$002D_{hex}$	Enable_All_Events_Request	8-39
$002E_{hex}$	Disable_All_Events_Request	8-40
0034_{hex}	Set_Parameter_Timeout_Constant_Request	8-26
0036_{hex}	Enable_Event_Logical_Address_Request	8-41
0037_{hex}	Disable_Event_Logical_Address_Request	8-42
$003A_{hex}$	Receive_Logical_IN_Address_Map_Request	8-14
$003B_{hex}$	Receive_Logical_OUT_Address_Map_Request	8-15
0040_{hex}	Implement_All_Logical_Address_Maps_Request	8-16
0047_{hex}	Enable_All_Messages_Request	8-26
0048_{hex}	Disable_All_Messages_Request	8-27
0049_{hex}	Receive_Group_Numbers_Request	8-9
$004A_{hex}$	Alarm_Stop_Request	8-18
$004C_{hex}$	Warmstart_Request	8-18
$004D_{hex}$	Set_Event_Message_Type_Request	8-43
$004E_{hex}$	Clear_Display_Request	8-19
0058_{hex}	Check_Physical_Configuration_Map_Request	8-8
0059_{hex}	Bus_Delay_Request	8-19
$005A_{hex}$	Send_Bus_Error_Information_Request	8-20
$005B_{hex}$	Send_Local_Bus_Module_Error_Request	8-21

Table 8-1: Commands for the IBS master board

Code	IBS command	Page
$005C_{hex}$	Send_All_Module_Error_Request	8-20
$005D_{hex}$	Receive_Processing_Instructions_Request	8-30
$005E_{hex}$	Send_Physical_Configuration_Request	8-6
$005F_{hex}$	Send_Log_Address_Error_Request	8-16
0060_{hex}	Define_Groups_Error_Characteristics_Request	8-12
0064_{hex}	Quit_Module_Error_Request	8-22
0065_{hex}	Quit_Module_Error_All_Request	8-23
0069_{hex}	Receive_Localbus_Code_Map_Request	8-13
$010D_{hex}$	Send_Actual_Configuration_Request	8-7
0114_{hex}	Read_Event_Counter_Request	8-44

8.1 Format of a Command Description

This section describes the commands for the IBS master board. The commands are given on a word basis, the command codes are in hexadecimal notation. The descriptions follow the following pattern.

Command nameCommand code$_{hex}$

Task: *Describes the functionality of the command.*

Prerequisite: *All conditions that are to be met before a command's call to ensure proper execution of the command.*

Syntax: Enter only the command code for a command without parameters.

The syntax of a command with parameters is displayed as a parameter block as follows:

Word 1	*Command code*
Word 2	*Number of subsequent words (parameter count)*
Word 3	*Parameter 1*
Word 4	*Parameter 2*
Word 5	*Parameter 3*
	...
Word n+2	*Parameter n*

Bit	15 0

Key: Parameter *Description of the individual parameters*

Positive acknowledgment: *Message indicating the successful execution of a command.*
Meaning: *Explanation of the message.*
Comment: *Further information on the message.*

Negative acknowledgment: *Message indicating an error in the execution of the command.*
Meaning: *Explanation of the error cause.*
Comment: *Further information on the message.*

8.2 Configuration Commands

Configure_Bus_Request0023$_{hex}$

Task: This command causes the IBS master board to read in the currently connected bus configuration and to save it as the *initial configuration* in the controller board RAM.

Prerequisite: The bus must be in the STOP state when this command is used.

Do not call the command while the system is in operation, as it will delete all definitions and lists stored on the controller board.

Effect: All definitions and lists stored on the controller board before will be deleted:
- Logical address lists
- Group definition
- Process data linkage
- Event definition

The configuration keeps the controller board busy for a certain time. This blocks a new command execution sequence for a certain period of time after the positive acknowledgment has been sent. The time period depends on the bus length.

Syntax: The commands consists only of one word, the command code (0023$_{hex}$). No further parameters follow.

Positive acknowledgment: Quit_Configure_Bus_Confirmation (00CA$_{hex}$)

Negative acknowledgment: Bus_System_Error_Indication (0038$_{hex}$)
Meaning: An error that allows no further operation has occurred.
Comment: The *Bus_Error_Information_Indication* (80C4$_{hex}$) message gives a detailed description of the error.

Send_Physical_Configuration_Request005E$_{hex}$

Task: This command reads out the bus configuration that is stored *as the initial configuration* in the controller board RAM in the form of length and ID codes. This configuration has been stored there by the execution of the *Configure_Bus_Request* command (0023$_{hex}$) or by the *Check_Physical_Configuration* command (0058$_{hex}$).

Comment: If the currently connected bus configuration is to be read out, execute the *Configure_Bus_Request* command (0023$_{hex}$) before calling the *Send_Physical_Configuration_Request* command.

Syntax: The command consists only of a single word, the command code 005E($_{hex}$). No further parameters follow.

Positive acknowledgment: Send_Physical_Configuration (80F4$_{hex}$)
Meaning: The acknowledgment transfers the bus configuration

stored in the controller board RAM in the form of length and ID codes.

Negative acknowledgment: No_Executable_Configuration ($004A_{hex}$)

Meaning: No bus system was connected during hardware startup (power on, configuration, or after the last reset).

Send_Actual_Configuration_Request$010D_{hex}$

Task: This command causes the IBS master board to transfer the currently connected bus configuration to the host with the *Send_Actual_Configuration_Confirmation* message (8119_{hex}).

Comment: Contrary to the *Send_Physical_Configuration_Request* command ($05E_{hex}$), not the bus configuration stored in the controller board RAM is transferred, but the actually connected configuration.

If the bus is in the STOP state at the moment when the command is called, the currently connected bus configuration cannot be transferred. Instead, the last configuration connected before the stop is transferred. The IBS master board will not recognize any configuration changes carried out after the bus has stopped! Only starting the bus or carrying out a new configuration with the *Configure_Bus_Request* (0023_{hex}) or *Check_Physical_Configuration* command (0058_{hex}) will cause these changes to be taken into account. If the bus goes into the STOP state due to a configuration change, the new (faulty) configuration will be transferred.

Syntax: The command consists only of a single word, the command code ($010D_{hex}$). No further parameters follow.

Acknowledgment: Send_Actual_Configuration_Confirmation (8119_{hex})

Meaning: The acknowledgment returns the currently connected bus configuration in the form of length and ID codes. When, for example, the remote bus cable is not connected to the controller board when the *Send_Actual_Configuration* command is called, this message will also be returned, but in this case without length and ID codes.

Check_Physical_Configuration_Map_Request0058$_{hex}$

Task: This command transfers a desired bus configuration to the controller board in the form of length and ID codes. This desired bus configuration is first checked whether the entered parameters are on principle valid (e.g. for valid length or ID codes) and, if they are valid, stored in the controller board RAM. Then the desired bus configuration is compared with the currently connected bus configuration.

Prerequisite: The bus must be in the STOP state when this command is used.

Syntax:

Word 1	Code		
Word 2	Parameter count (n)		
Word 3	Length code	ID code	for device 1
Word 4	Length code	ID code	for device 2
Word 5	Length code	ID code	for device 3
	...	...	
Word n+2	Length code	ID code	for device n
Bit	15 8	7 0	

Key:

Code: Command code (here 0058$_{hex}$)

Parameter count: Number of subsequent words (here: number of devices).

Length code: The length code describes the address space requirement of the IBS device in the host.

ID code: Identification code of the IBS device. It is printed on the modules as *Module ID* in decimal notation.
(0 to 255$_{dec}$ corresponds to 00$_{hex}$ to FF$_{hex}$)

After using the command, reload all address lists, group and event definitions! The length code for PCP devices contains only the number of process input/output data; the data to be used by PCP are not to be specified. Example: The *IBSV.24* module has only one word for PCP and no process input/output data. Therefore, enter a 0 for the length code of the module *IBSV.24*.

Positive acknowledgment: Physical_Configuration_Map_Valid_Confirmation (00AB$_{hex}$)

Meaning: The desired bus configuration is valid and has been stored in the controller board RAM. In addition, it matches the currently connected bus configuration.

Negative acknowledgment: Check_Configuration_Error_Confirmation (0068$_{hex}$)

Meaning: The desired bus configuration is valid and has been stored in the controller board RAM. However, it does **not** match the currently connected bus configuration.

Negative acknowledgment: Unknown_Bus_Module ($005B_{hex}$)

Meaning: An invalid number of process data words was specified; the first module of the bus configuration is no bus terminal module.

Comment: Only in this case is the desired configuration not stored in the controller board RAM.

Receive_Group_Numbers_Request0049_{hex}

Task: Using this command, bus segments that belong together logically are combined into one group, by specifying the group number in connection with the bus segment numbers. Assign all bus segments, with the exception of the segments in installation remote busses, to one group! In the basic condition (following power on, pressing the reset button or calling the *Configure_Bus_Request* (0023_{hex})), all bus segments are combined in group 0.

Prerequisite: The bus must be in the STOP state when this command is used.

Comment: The distribution of the bus configuration into groups makes it possible to disable and reenable the bus segments that branch off from the main line (remote bus branch line, installation remote bus, local bus; commands: *Switch_Group_Off_Request* and *Switch_Group_On_Request*). A group may consist of one or more bus segments.

Syntax:

Word 1	Code		
Word 2	Parameter count (n)		
Word 3	00_{hex}	Bus segment	for 1st bus segm.
Word 4	00_{hex}	Group	for 1st bus segm.
Word 5	00_{hex}	Bus segment	for 2nd bus segm.
Word 6	00_{hex}	Group	for 2nd bus segm.
	...	...	
	...	...	
Word n+1	00_{hex}	Bus segment	for xth bus segm.
Word n+2	00_{hex}	Group	for xth bus segm.

Bit	15 8	7 0

Key:

Code: Command code (here 0049_{hex})

Parameter count: Number of subsequent words (here: number of bus segments multiplied by 2)

Bus segment: Number of the bus segment assigned to the group mentioned in the next line

Group: Number of the group to which the bus segment mentioned in the previous line is assigned. (0 to 255_{dec} corresponding to 00_{hex} to FF_{hex})

Assign a group number from 0 to 255 to all bus segments (BS), with the exception of the segments in the branch lines (RB level 1).

- Assign separate group numbers to all remote bus devices (e.g. bus terminal modules with I/O function).
- IBS devices without branching-off lines (no local bus connection, no remote bus branch) also get a group number of their own!
- According to the type of addressing, enter the logical or physical bus segment number in the *bus segment* parameter.
- Bus segment numbers may be assigned only once!
- Enter only IBS devices that are in the main line of the bus configuration (RB level 0). Devices in the remote bus branch or in the local bus (RB level 1) are included in the group definition by assigning the bus terminal module at the beginning of this branch line to a group.

Positive acknowledgment: Quit_Receive_Group_Numbers_Confirmation ($00BD_{hex}$)

Negative acknowledgment: Receive_Group_Numbers_Failed_Confirmation (0055_{hex})
Meaning: An error was detected when the group was checked.

Switch_Group_Off_Request0021_{hex}

Task: The command disables a previously defined group of bus segments. The outputs of the IBS devices in the disabled bus segment and the associated inputs of the host control system (or computer system) are set to 0.

Prerequisite: Before this command is called, a group definition (*Receive_Group_Numbers_Request* 0049_{hex}) must have been carried out.

The instructions of process data links which work with addresses within the disabled groups are no longer active. Using the application program, ensure that the system state allows this. The input process data is set to zero.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Group number

Bit	15 8	7 0

Key:
Code: Command code (here 0021_{hex})
Parameter count: Number of subsequent words (here 1)
Group number: Number of the group to be disabled.
(0 to 255_{dec} corresponds to 00_{hex} to FF_{hex})

When a system part is disabled, the supply function for the module electronics (logic voltage) must be retained for the included IBS devices with bus terminal module functionality to ensure that the operation of the remaining bus configuration can continue.

Positive acknowledgment: Switch_Group_Off_Confirmation ($809D_{hex}$)

Negative acknowledgment: Not_Expected_Group_Number (0059_{hex})
Meaning: Unknown or invalid group numbers; attempt to disable a module that cannot be disabled

Switch_Group_On_Request0020_{hex}

Task: The command enables a previously defined group of bus segments. The enabled segments are not checked.

Prerequisite: Ensure that the group definition (*Receive_Group_Numbers_Request* (0049_{hex}) has taken place before this command is called.

The process outputs used in the bit manipulation are included in the process data transmission again when a group is reenabled. Using the application program, ensure that this data is set to the correct state.

Positive acknowledgment: Switch_Group_On_Confirmation ($809E_{hex}$)
Comment: The bus system continues to operate with the enabled groups. If an enabled group is defective, the error messages *Bus_System_Error_Indication* (0038_{hex}) and *Bus_Error_Information_Indication* ($80C4_{hex}$) may follow.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Group number

Bit	15 8	7 0

Key:
Code: Command code (here 0020_{hex})
Parameter count: Number of subsequent words (here 1)
Group number: Number of the group to be enabled.

Negative acknowledgment: Not_Expected_Group_Number_Indication (0059_{hex})
Meaning: An unknown group number was used, or the number is not in the valid range.

Negative acknowledgment: Switch_Group_On_Failed_Confirmation ($80C5_{hex}$)
Meaning: The group cannot be disabled as the configuration was changed or modules in the main line (RB level 0) are missing.

Define_Groups_Error_Characteristic_Request0060$_{hex}$

Task: If an error which does not allow any further bus operation occurs in a group, the bus will go into the RESET mode. Using the *Define_Group_Error_Characteristics_Request* you can define for each group whether the bus is to be started again without this group after the RESET.

Prerequisite: Ensure that a group definition has taken place with the (*Receive_Group_Numbers_Request* command (0049$_{hex}$) before this command is used.

Syntax:

Word 1	Code			
Word 2	Parameter count			
Word 3	Mode	0_{hex}	Group number	for group 1
Word 4	Mode	0_{hex}	Group number	for group 2
	...	...	...	
Word n+2	Mode	0_{hex}	Group number	for group n

Bit	15 12	11 8	7 0

Key:

Code: Command code (here 0060$_{hex}$)

Parameter count: Number of subsequent words

Mode: Handling of the group described by this line in the event of an error:

- 0_{hex} Following a RESET caused by this group, the complete bus remains in the STOP state.
- 8_{hex} Following RESET, the bus is restarted without this group.

Group number: Group number of the group described by this line (0 to 255$_{dec}$ corresponds to 00$_{hex}$ to FF$_{hex}$)

You need not specify all numbers. If an error occurs in a group which you did not specify, the complete bus remains in the STOP state after the RESET.

The controller board does not supply new input data to the host while the error diagnostics are being executed. The message *Bus_Error_Information_Indication* (80C4$_{hex}$) gives a detailed description of the error.

Positive acknowledgment: Quit_Groups_Error_Characteristics_Confirmation (00F6$_{hex}$)

Negative acknowledgment: Groups_Error_Characteristics_Failed_Confirmation (80F7$_{hex}$)

Comment: In the event of a negative acknowledgment, the definitions that were made with this command before and were positively acknowledged remain valid!

8.3 Addressing Commands

Receive_Localbus_Code_Map_Request0069$_{hex}$

Task: The command carries out the selectable assignment of the bus segment number to the InterBus-S devices with bus terminal module functionality (remote bus and installation remote bus devices). This is recommended so that the numbering of the bus segments in the existing system does not need to be changed when the system is expanded.

Prerequisite: None

Syntax:

Word 1	Code			
Word 2	Parameter count			
Word 3	RB level	0_{hex}	Bus segment number	Device 1
Word 4	RB level	0_{hex}	Bus segment number	Device 2
	...	...	...	
Word n+2	RB level	0_{hex}	Bus segment number	Device n

Bit	15 12	11 8	7 0

Key:

Code: Command code (here 0069_{hex})

Parameter count: Number of subsequent words (here: number of bus segments).

RB level: Enter here the RB level:
- Main line: RB level = 0_{hex} (0000_{bin})
- Remote bus branch line (e.g. branching off from IBS IP CBK): RB level = 1_{hex} (0001_{bin})

Bus segment number: Enter here the desired logical bus segment numbers in the order of their physical arrangement (0 to 255_{dec} corresponds to 00_{hex} to FF_{hex}). Every address may be assigned only once!

The list is only valid after the *Implement_All_Logical_Address_Maps_Request* command (0040_{hex}) has been executed successfully.

Positive acknowledgment: Quit_Receive_Localbus_Code_Map_Confirmation (0105_{hex})

Receive_Logical_IN_Address_Map_Request003A$_{hex}$

Task: The command transfers the list of logical input addresses to the controller board.

Syntax:

Word 1	Code				
Word 2	Parameter count				
Word 3	DC	0000_{bin}	Input address		Device 1
Word 4	DC	0000_{bin}	Input address		Device 2
	...	...	...		
Word n+2	DC	0000_{bin}	Input address		Device n

Bit	15 14	13 9	9 0

Key:

Code: Command code (here $003A_{hex}$)

Parameter count: Number of subsequent words (here number of devices).

Input address: Enter here (10 bits) the desired address (as a byte address) for all modules with process input data. With modules with more than one process input word (e.g. with 32 bits), the next higher addresses are automatically also assigned.

DC: Enter here the data consistency for the process data access.The data consistency ensures that the specified data width is from the same IBS cycle. The default value for I/O modules is 16 bits (00_{bin}). Increase the data consistency for IBS devices which require the coherent transmission of larger data widths. Such devices are, for example, encoders, operator interfaces or analog modules with a resolution of more than 16 bits. The following values are permissible for the data consistency:

- 00_{bin} 16 bits (default)
- 01_{bin} 32 bits (e.g. encoders, operator interfaces)
- 10_{bin} 8 bits (only for modules with a length code of 81_{hex}, to which a byte access is to be made!)
- 11_{bin} 48 bits (e.g. encoders, operator interfaces)

Enter 0000_{hex} for all modules without process input data (e.g. dedicated bus terminal modules or dedicated output modules).

Place the start addresses of IBS devices with an address area of 16 bits (or more) only at even addresses.

Positive acknowledgment: Quit_Receive_Logical_In Address_Map_ Confirmation ($00D1_{hex}$)

The list will only be valid after the successful execution of the *Implement_All_Logical_Address_Maps_Request* (0040_{hex}) command.

Receive_Logical_OUT_Address_Map_Request003B$_{hex}$

Task: The command transfers the list of logical output addresses to the controller board.

Syntax:

Word 1	Code			
Word 2	Parameter count			
Word 3	DC	0000_{bin}	Output address	Device 1
Word 4	DC	0000_{bin}	Output address	Device 2
	...	...	...	
Word n+2	DC	0000_{bin}	Output address	Device n

Bit	15 14	13 9	9 0

Key:

Code: Command code (here 003B$_{hex}$)

Parameter count: Number of subsequent words (here number of devices).

Output address: Enter here (10 bits) the desired address (as byte address) for all modules with process output data. With modules with more than one process output word (e.g. with 32 bits) the next higher addresses are automatically also assigned.

DC: See the parameter description for the *Receive_Logical_IN_Address_Map_Request*command (003A$_{hex}$)

Enter 0000$_{hex}$ for all modules without process output data (e.g. dedicated bus terminal modules or dedicated input modules).

☞ Place the start addresses of IBS devices with an address area of 16 bits (or more) only at even addresses.

Positive acknowledgment: Quit_Receive_Logical_Out Address_Map_ Confirmation (00D2$_{hex}$)

☞ The list will only be valid after the successful execution of the *Implement_All_Logical_Address_Maps_Request* (0040$_{hex}$) command.

Implement_All_Logical_Address_Maps_Request0040$_{hex}$

Task: The controller board checks the previously transferred address lists and accepts them if they do not contain errors.

The command may only be used while the bus data cycle is not active! The previously sent lists are checked and accepted if they do not contain errors. (In the event of an error you can inquire the error causes with the *Send_Log_Address_Error_Request* (005F$_{hex}$) command.)

Syntax: The command consists only of a single word, the command code 0040($_{hex}$). No further parameters follow.

Positive acknowledgment: Quit_Implement_Confirmation (00D3$_{hex}$)
Meaning: The addresses are valid and have been accepted.

Negative acknowledgment: Logical_Address_Error_Confirmation (002B$_{hex}$)
Meaning: The lists to be implemented (logical addresses) contain faulty values. The bus is in the STOP state. The process input data is **not** switched into a defined, safe state!

Send_Log_Address_Error_Request005F$_{hex}$

Task: The results of a previous execution of one of the following commands is requested:
- Implement_All_Logical_Address_Maps_Request (0040$_{hex}$)
- Check_Physical_Configuration_Request (0058$_{hex}$)

Syntax: The command consists only of a single word, the command code (005F$_{hex}$). No further parameters follow.

Positive acknowledgment: Send_Log_Address_Error_Confirmation (80F5$_{hex}$)
Meaning: The result is announced with this message (see Section 9).

8.4 Operation Commands

Start_Bus_Cycle_Request0001$_{hex}$

Task: The command activates the cyclic data traffic on the bus. The controller board puts the bus configuration into operation and provides the I/O data in the MPM.

Syntax: The command consists only of a single word, the command code (0001$_{hex}$). No further parameters follow.

Positive acknowledgment: Start_Bus_Confirmation (0088$_{hex}$)
Meaning: The system could be started. The controller board starts to cyclically read in and set the inputs and outputs (process data and possibly PCP mode).

Comment: The *Masterboard RUN* LED is on.

Negative acknowledgment: Start_Bus_Not_Possible_Confirmation ($00E3_{hex}$)

Meaning: The maximum permissible bus configuration has been exceeded,
all groups are disabled,
controller board hardware error,

Comment: The *RUN Masterboard* LED is not on.

Negative acknowledgment: Bus_System_Error_Indication (0038_{hex})

Meaning: An error which allows no further bus operation has occurred.

Comment: The *Bus_Error_Information_Indication* message ($80C4_{hex}$) gives a detailed description of the error.

The error message *Module_Error_Indication* ($80A0_{hex}$), which may follow a positive or negative acknowledgment, does not cause data traffic to abort. When the bus configuration is changed before the start of the data traffic, the data traffic first starts and is aborted immediately afterwards.

Stop_Bus_Cycle_Request0002_{hex}

Task: This command switches the bus into the STOP state.

Consequences: Process data channel: The cyclic data traffic on the bus is stopped. The existing process data image of the modules connected to the bus is statically retained and will not be updated.

PCP channel: The Peripherals Communication Protocol (PCP) is not operated any further. **No** automatic abort of the established connections takes place. The processing of any outstanding services will continue after the restart of the data traffic.

This command does **not** switch the process output data into the safe state (reset of the outputs). This can only be done with the *Alarm_Stop_Request* command ($004A_{hex}$).

Syntax: The command consists only of a single word, the command code (0002_{hex}). No further parameters follow.

Positive acknowledgment: Stop_Bus_Confirmation ($00C6_{hex}$)

Meaning: The operation of the process data channel and, if in use, of the communication channel (PCP) is discontinued.

Comment: The *RUN Masterboard* LED goes out.

Alarm_Stop_Request004A$_{hex}$

Task: The command causes a reset and switches the bus into the STOP state. The command is executed directly after the completion of the current data cycle.

Consequences:
Process data channel: The cyclic data traffic of the bus is stopped. The existing process data image of the modules connected to the bus (host system inputs) is set to 0. The command switches the process output data (outputs of the IBS devices) into the safe state (to 0).
PCP channel: The Peripherals Communication Protocol (PCP) is not operated any further. **No** automatic abort of the established connections takes place. The processing of any outstanding services will continue after the restart of the data traffic.

Further operation: The following definitions and lists will be deleted on the controller board:
- logical address lists
- group definition
- process data linkage
- event definition

Syntax: The command consists only of a single word, the command code (004A$_{hex}$). No further parameters follow.

Positive acknowledgment: Quit_Alarm_Stop_Confirmation (00D8$_{hex}$)
Meaning: The bus data traffic has been stopped.
Comment: The parameter lists and the *Run* LED are cleared.

Warmstart_Request004C$_{hex}$

Task: This command carries out a warm start of the firmware. Following the hardware startup, the controller board hardware and software is initialized. All previously made settings become invalid and reset to the same state as if the reset button had been pressed.

Consequences: A reset is carried out on the bus. With modules with process data, this causes process data to be reset to the value 0. The bus data traffic is stopped.

The controller board is only ready again when the *Ready Masterboard* LED is on!

Syntax: The command consists only of a single word, the command code (004C$_{hex}$). No further parameters follow.

Acknowledgment: The execution of the command is not acknowledged. The *Ready Masterboard* LED first goes out and comes on again after the completion of the warmstart.

Bus_Delay_Request0059$_{hex}$

Task: When the bus system is operated with communication (PCP channel), there is a delay of 1 ms between the data cycles. The time is enabled during the initialization of the communication layer with the *Init_Comm_Service_Request* command (0054$_{hex}$). When there are no devices with communication functions in the bus system, the delay is automatically disabled. Using the *Bus_Delay_Request* command (0059$_{hex}$), the delay can also be disabled for the operation with communication.

Syntax: The command consists only of a single word, the command code 0059($_{hex}$). No further parameters follow.

Acknowledgment: Quit_Bus_Delay_Confirmation (00EC$_{hex}$).

8.5 Error Handling Commands

Clear_Display_Request004E$_{hex}$

Task: This command clears the status indicator on the controller board front plate. The entries in the diagnostic bit register and in the diagnostic parameter register are also reset.

Prerequisite: None

Syntax: The command consists only of a single word, the command code (004E$_{hex}$). No further parameters follow.

Positive acknowledgment: Quit_Clear_Display_Confirmation (00E2$_{hex}$)

Send_Bus_Error_Information_Request005A$_{hex}$

Task: This command requests the *Bus_Error_Information_Indication* message (80C4$_{hex}$). This message transfers a list with information on the bus error state (Bus Error Information Map) from the RAM. The list was generated at the last occurrence of a remote bus error or of a local bus error.

Prerequisite: Use this command only after the error message *Bus_System_Error_Indication* (0038$_{hex}$) has been sent. There is no check whether the bus error state has changed again since the generation of the *Bus Error Information Map* (e.g. the user has already corrected the error) and, therefore, the contents of the last *Bus Error Information Map* are no longer up to date.

Syntax: The command consists only of a single word, the command code (005A$_{hex}$). No further parameters follow.

Positive acknowledgment: Bus_Error_Information_Indication (80C4$_{hex}$)
Meaning: List with information on the error state of the bus (Bus Error Information Map)

Negative acknowledgment: No_Map_Entry_Confirmation (00ED$_{hex}$)
Meaning: There is no entry.

Send_All_Module_Error_Request005C$_{hex}$

Task: When the bus system is active, this command checks all modules for error messages and updates the module error indication in the diagnostic interface.

Prerequisite: Module error

Syntax: The command consists only of a single word, the command code (005C$_{hex}$). No further parameters follow.

Positive acknowledgment: Send_All_Module_Error_Confirmation (80EF$_{hex}$)
Meaning: This acknowledgment sends a list of all bus segments with module errors.

Negative acknowledgment: No_Map_Entry_Confirmation (00ED$_{hex}$)
Meaning: There are no module errors. The error indications in the diagnostic interface are reset.

Send_Local_Bus_Module_Error_Request005B$_{hex}$

Task: This command looks in the specified local bus for modules indicating a module error.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Bus segment

Bit	15 8	7 0

Key:

Code:	Command code (here $005B_{hex}$)
Parameter count:	Number of subsequent words (here 1).
Bus segment:	Enter here the bus segment number of the local bus where modules with module errors are to be searched for.

Positive acknowledgment: Local_Bus_Module_Error_Confirmation ($80EE_{hex}$)

Meaning: A list with positions and ID codes of all modules where a module error has occurred. The position is the physical location number in the specified local bus (see also Section 9):
- Bus terminal modules: 0
- Local bus devices: 1 to 8

Negative acknowledgment: No_Map_Entry_Confirmation ($00ED_{hex}$)

Meaning: No module errors were found in the specified local bus.

Quit_Module_Error_Request0064$_{hex}$

Task: IBS devices with indication hold feature indicate a module error also after it has been corrected. Therefore, the error message must be acknowledged. The command acknowledges the module errors of the specified IBS devices. Specify the bus segment number and the location in the local bus for each IBS device to be acknowledged.

Prerequisite: Module errors on an IBS device with indication hold feature.

Syntax:

Word 1	Code		
Word 2	Parameter count		
Word 3	00_{hex}	Bus segment	Device 1
Word 4	00_{hex}	Location	Device 1
Word 5	00_{hex}	Bus segment	Device 2
Word 6	00_{hex}	Location	Device 2
	...	...	
	...	...	
Word n+1	00_{hex}	Bus segment	Device x
Word n+2	00_{hex}	Location	Device x

Bit	15 8	7 0

Key:

Code: Command code (here 0064_{hex})

Parameter count: Number of subsequent words (here number (x) of devices for which a module error message is to be acknowledged, multiplied by 2).

Bus segment: Logical bus segment number (0 to 255_{dec} corresponds to 00_{hex} to FF_{hex})

Location: Physical position number in the bus segment selected by the previous word:

- bus terminal modules, dedicated remote bus devices and installation remote bus devices: 0;
- Local bus devices: 1 to 8

Positive acknowledgment: Quit_Module_Error_Ok_Confirmation ($00FE_{hex}$)

Meaning: The command has been executed successfully.

Comment: The error LEDs on the IBS have also been reset.

Negative acknowledgment: Quit_Module_Error_Not_Possible_Confirmation ($80FF_{hex}$)

Quit_Module_Error_All_Request0065$_{hex}$

Task: This command acknowledges the module error messages of all IBS devices with indication hold feature in the currently accessible bus configuration.

Prerequisite: Module error auf IBS devices with indication hold feature

Comment: The error LEDs on the IBS devices are also cleared.

Syntax: The command consists only of a single word, the command code (0065_{hex}). No further parameters follow.

Positive acknowledgment: Quit_Module_Error_Ok_Confirmation ($00FE_{hex}$)
Meaning: The command has been executed successfully.

Negative acknowledgment: Quit_Module_Error_Not_Possible_Confirmation ($80FF_{hex}$)
Meaning: The module error message was not acknowledged for all accessible IBS devices with indication hold feature.
Comment: For each IBS device whose module error could not be acknowledged, the message contains two parameters with a more detailed description of the cause (see description of the message ($80FF_{hex}$).

Set_BK_Alarm_Logical_Request0024$_{hex}$

Task: The command sets the alarm output of the BK module.

Prerequisite: There must be an alarm output on the accessed BK module.

If the accessed module has no alarm output, a positive acknowledgment is generated!

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Log. bus segment

Bit	15 8	7 0

Key: Code: Command code (here 0024_{hex})
Parameter count: Number of subsequent words (here 1).
Log. bus segment: Enter here the logical bus segment number (0 to 255_{dec} corresponds to 00_{hex} to FF_{hex}) of the bus terminal module whose alarm output you want to set.

Positive acknowledgment: Quit_Set_BK_Alarm_Logical_Confirmation ($00CB_{hex}$)

Neg. acknowl.: BK_Alarm_Failed_Confirmation ($005A_{hex}$)
Meaning: Invalid local bus address.

Reset_BK_Alarm_Logical_Request0025$_{hex}$

Task: The command resets the alarm output of a BK module.

Prerequisite: There must be an alarm output on the accessed BK module, and it must have been set.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Log. bus segment

Bit	15 8	7 0

Key:

Code:	Command code (here 0025_{hex})
Parameter count:	Number of subsequent words (here 1).
Log. bus segment:	Enter here the logical bus segment number (0 to 255_{dec} corresponds to 00_{hex} to FF_{hex}) of the bus terminal module whose alarm output you want to reset.

Positive acknowledgment: Quit_Reset_BK_Alarm_Logical_Confirmation ($00CC_{hex}$)

Negative acknowledgment: BK_Alarm_Failed_Confirmation ($005A_{hex}$)
Meaning: Invalid local bus address or too many parameters.

Set_BK_Alarm_Physical_Request0026$_{hex}$

Task: The command sets the alarm output of a BK module.

Prerequisite: There must be an alarm output on the accessed BK module.

If the accessed module has no alarm output, the positive acknowledgment is generated!

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Phys. bus segment

Bit	15 8	7 0

Key:

Code:	Command code (here 0026_{hex})
Parameter count:	Number of subsequent words (here 1).
Phys. bus segment:	Enter here the physical bus segment number (0 to 255_{dec} corresponds to 00_{hex} to FF_{hex}) of the bus terminal module whose alarm output you want to set.

Positive acknowledgment: Quit_Set_BK_Alarm_Physical_Confirmation ($00CD_{hex}$)

Negative acknowledgment: BK_Alarm_Failed_Confirmation ($005A_{hex}$)
Meaning: Invalid local bus address.

Reset_BK_Alarm_Physical_Request0027_{hex}

Task: The command resets the alarm output of a bus terminal module.

Prerequisite: There must be an alarm output on the accessed bus terminal module, and it must have been set.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Phys. bus segment

Bit	15 8	7 0

Key:
Code: Command code (here 0027_{hex})
Parameter count: Number of subsequent words (here 1).
Phys. bus segment: Enter here the physical bus segment number (0 to 255_{dec} corresponds to 00_{hex} to FF_{hex}) of the bus terminal module whose alarm output you want to reset.

Positive acknowledgment: Quit_Reset_BK_Alarm_Physical_Confirmation ($00CE_{hex}$)

Negative acknowledgment: BK_Alarm_Failed_Confirmation ($005A_{hex}$)
Meaning: Invalid local bus address or too many parameters.

8.6 Application Interface Commands

Set_Parameter_Timeout_Constant_Request0034$_{hex}$

Task: This command determines the time constant for the command and message time-out.

The controller board monitors the exchange of commands and messages between the host and the controller board with a timeout feature. When a message is not fetched within the timeout period, or if the internal message memory overflows, the controller board generates the error message *Controller_Parameter_Timeout_Indication* (000C$_{hex}$) without regard of the transmission protocol.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	Reserved	
Word 4	00_{hex}	Timeout

Bit	15 8	7 0

Key:

Code:	Command code (here 0034$_{hex}$)
Parameter count:	Number of subsequent words (here 2)
Reserved:	This parameter is reserved for future additions. Enter the value 0000$_{hex}$.
Timeout:	Enter here the desired timeout period in milliseconds.

Positive acknowledgment: Quit_Set_Parameter_Timeout_Confirmation (00CF$_{hex}$)

Enable_All_Messages_Request0047$_{hex}$

Task: This command enables the transfer of messages from the controller board to the host.

The transfer of messages is enabled until the user disables it. Following RESET it is disabled as default!

Syntax: The command consists only of a single word, the command code (0047$_{hex}$). No further parameters follow.

Positive acknowledgment: Quit_Enable_All_Messages_Confirmation (00D6$_{hex}$)

Disable_All_Messages_Request0048$_{hex}$

Task: The command disables the transfer of messages from the controller board to the host and routes them to the diagnostic interface (V24).

The *Enable_All_Messages_Request* (0047$_{hex}$) command cancels the redirection.

Syntax: The *Disable_All_Messages_Request* command consists only of a single word, the command code (0048$_{hex}$). No further parameters follow.

Positive acknowledgment: Quit_Disable_All_Messages_Confirmation (00D6)

8.7 System Check Commands

Send_Software_Revision_Request0008$_{hex}$

Task: The command inquires the firmware version on the IBS master.

Prerequisite: None

Syntax: The command consists only of a single word, the command code (0008$_{hex}$). No further parameters follow.

Positive acknowledgment: Software_Revision_Confirmation (8087$_{hex}$)

Send_Bus_Cycle_Counter_Request0006$_{hex}$

Task: A 32-bit counter that continuously counts the InterBus-S data cycles runs internally on the controller board. This counter is set to 0 when the controller board starts up. The command reads out the count.

Syntax: The command consists only of a single word, the command code (0006$_{hex}$). No further parameters follow.

Positive acknowledgment: Bus_Cycle_Counter_Confirmation
Meaning: The message indicates the current counter reading.

8.8 Process Data Linkage Commands

In some applications, input signal transitions should be responded to faster than the host system is able to. The host response time to a signal transition depends on the application program cycle time. Process data linkage makes it possible to respond to signal transitions at inputs before they are included in the process image of the host. When the controller board detects the signal transition of an input, it can

- pass it on in the usual way to the host for further processing,
- quickly effect a direct response to an output using a process data linkage instruction.

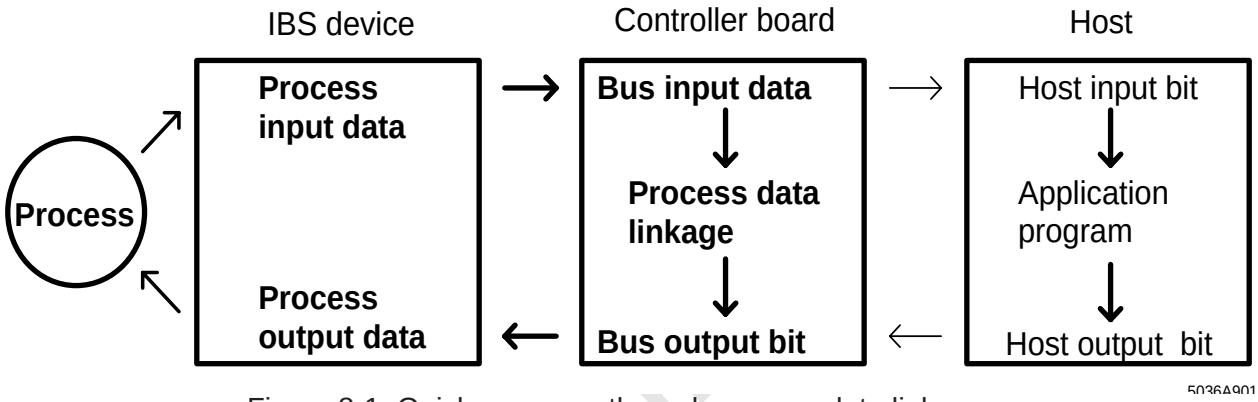


Figure 8-1: Quick response through process data linkage

In connection with process data linkage please use logical addressing as far as possible. Process data linkage can on principle also be done under physical addressing, but this would mean that if you want to expand your system later on you would have to adapt all address specifications in all assignment lists. Under logical addressing you only need to add the new IBS devices to the assignment lists.

General functions of the process data linkage instructions

The instructions control selectable process output data bits (bus output bits) on the basis of selectable process input data bits (bus input bit) and the output data written by the host (host output bit).

The representation of the bits in the bus input word in Figure 8-2 is valid for:

a) instructions which cause the bus output bits to reset on the occurrence of the positive edge of the bus input bit.
b) instructions which cause the bus output bits to reset on the occurrence of the negative edge of the bus input bit.

The bit in the bit manipulation register is only relevant with the instructions for conditional process data linkage
(*Reset_Out_In01_Switch* and *Reset_Out_In10_Switch*).

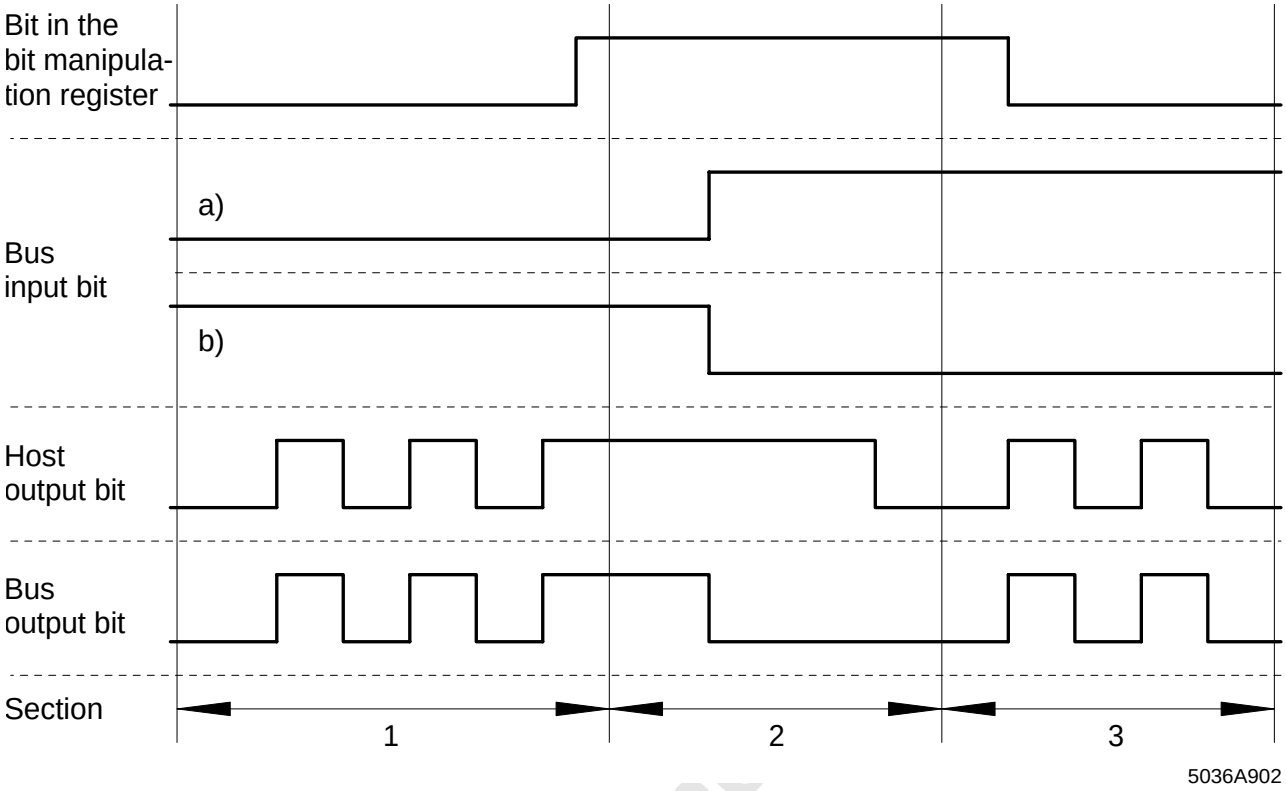


Figure 8-2: Pulse diagram for process data linkage

General function of the RESET... instructions

The bus output bit is reset following the evaluation of the bus input bit edge (section 2 of the diagram above). The instruction determines the evaluation of the bus input bit. The bus output bit is set via the setting of the host output bit (sections 1 and 3 in the diagram above). As long as the bus output bit has not been reset by the evaluation of the bus input bit, the bus output bit is determined only by the host output bit (sections 1 and 3 in the diagram above).

Table 8-2: Process data linkage instructions

Code	Instruction	Comment	Page
0000_{hex}	Clear_All_Processing_ Instructions	Deletes all instructions stored before	8-32
0005_{hex}	Reset_Out_In01	Responds to positive edge (0→1)	8-32
0006_{hex}	Reset_Out_In10	Responds to negative edge (1→0)	8-32
0007_{hex}	Reset2_Out_In01	Responds to positive edge (0→1)	8-33
0008_{hex}	Reset2_Out_In10	Respons to negative edge (1→0)	8-34
0009_{hex}	Reset2_Out_In01_Switch	Responds to positive edge (0→1)	8-35
$000A_{hex}$	Reset2_Out_In10_Switch	Responds to negative edge (1→0)	8-36
$000D_{hex}$	Bit_Copy	Copies an input bit to an output bit	8-37
$000E_{hex}$	Word_Copy_Bit_Mask	Linkage of an input word with a mask	8-37

Process data linkage can only be carried out in the address area of the binary input and outputs. The cycle time of the data traffic on the bus may increase by up to 40 µs per instruction. The instructions are transferred to the controller board with the *Receive_Processing_Instructions_Request* command ($005D_{hex}$).

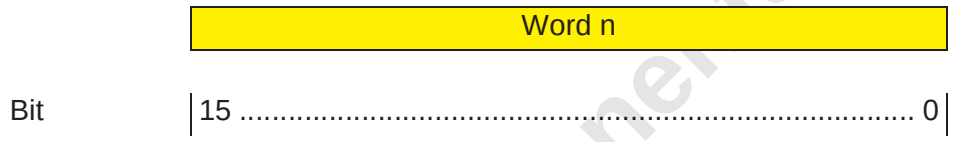
Address specifications within the process data linkage instructions

Specify the input bit and the output bit for each process data linkage procedure. Address these bits by specifying the address and the bit number.

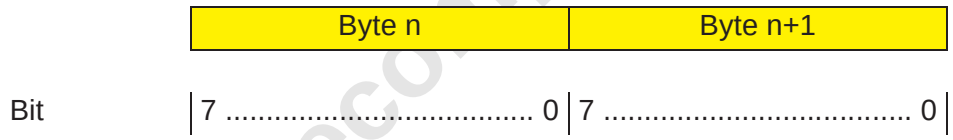
- Specify the even word address and the desired bit (0 to 15) for IBS devices with an address area of 16 bits (or more)
- Specify the even or odd byte address and the desired bit (0 to 7) for IBS devices with an address area of 8 bits (length code 81_{hex})

The word and byte numbering typical for InterBus is explained in Section 6.3.4.

A word of an IBS device with an address area of 16 bits (or more):



A word with two IBS devices, each with an address area of 8 bits:



According to the addressing mode, use the physical or the logical address, which you assigned with the commands
- *Receive_Logical_IN_Address_Map_Request* ($003A_{hex}$)
- *Receive_Logical_OUT_Address_Map_Request* ($003B_{hex}$).

Receive_Processing_Instructions_Request$005D_{hex}$

Task:

The *Receive_Processing_Instructions_Request* ($005D_{hex}$) command transfers one or more process data linkage instructions to the controller board. The instructions are sent to the controller board in the form of a list of instructions, and stored in the controller board RAM.

Transfer the lists of instructions before the start of the IBS system and - if used - following the command *Implement_All_Logical_Address_Maps_Request*. The process data link must **not** be redefined during the operation and must not be used in connection with group definitions, as this may cause bits of the process output data used by the process data linkage to be reset.

A memory which takes up to 1000 entries is available for the list of instructions. This corresponds to, e.g.:

- 332 three-word instructions (instruction code 0005_{hex} or 0006_{hex})
- 199 five-word instructions (instruction code 0007_{hex} or 0008_{hex})
- 142 seven-word instructions (instruction code 0009_{hex} or $000A_{hex}$)

Instructions with invalid addresses are not executed. There is **no** separate message for this situation. Addresses (and, therefore, instructions) become invalid with

- logical addressing when previously used addresses no longer exist, or
- disabled groups when the address is in the area of a disabled group.

Syntax:

Word 1	Code	
Word 2	Parameter count (n)	
Word 3	Instruction code	
Word 4	1st instruction parameter	
Word 5	2nd instruction parameter	1st instruction
etc.	...	
	Last instruction parameter	
	Instruction code	
	1st instruction parameter	
	2nd instruction parameter	2nd instruction
	...	
	Last instruction parameter	
	...	
	...	
	Instruction code	
	1st instruction parameter	
	2nd instruction parameter	xth instruction
	...	
Word n+2	Last instruction parameter	
Bit	15 0	

Key:

Code: Command code (here $005D_{hex}$)

Parameter count: Number of subsequent words (here: total of the individual instruction lengths).

Positive acknowledgment: Quit_Receive_Instructions_Confirmation (00F2)

Meaning: The parameters are error-free. All executed instructions have been accepted and added to a list, if such a list already exists.

Negative acknowledgment: Processing_Instructions_Error_Confirmation (80F3)
Meaning: The parameters are faulty. The sent list is not accepted (see Section 9).

Clear_All_Processing_Instructions Instruction (0000_{hex})

Task: This instruction deletes all instructions in the controller board RAM. Set the instruction to the beginning of an instruction list to be transferred if the instruction list is to overwrite the instructions that were stored in the RAM before. If you leave out the *Clear_Instruction_List* instruction, the newly transferred instructions will be added to the already stored ones.

Comment: Transfer the instruction to the controller board using the command *Receive_Processing_Instructions_Request* ($005D_{hex}$).

Syntax: The instruction consists only of a single word, the instruction code (0006_{hex}). No further parameters follow.

Reset_Out_In01 Instruction (0005_{hex})

Task: This instruction causes a particular bit in the specified output word to be reset when the **positive** edge of a particular bit in the specified input word is detected.

Transfer the instruction to the controller board using the command *Receive_Processing_Instructions_Request* ($005D_{hex}$).

Syntax:

Word 1	Code		
Word 2	Bit number	res.	Output address
Word 3	Bit number	res.	Input address

Bit	15 12	11	10 0

Key:
Code: Instruction code (here 0005_{hex})
Bit number: Enter here the number of the desired bit (0_{hex} to F_{hex} corresponds to 0_{dec} to 15_{dec}).
Res.: This bit is reserved for future extensions. Set it to 0.
Output address: Address of the word with the bit manipulated by the bit manipulation
Input address: Address of the word with the bit determining the bit manipulation

Instruction *Reset_Out_In10* (0006_{hex})

Task: This instruction causes a particular bit in the specified output word to be reset when the **negative** edge of a particular bit in the specified input word is detected.

Transfer the instruction to the controller board using the command *Receive_Processing_Instructions_Request* ($005D_{hex}$).

Syntax:

Word 1	Code		
Word 2	Bit number	res.	Output address
Word 3	Bit number	res.	Input address

Bit	15 12	11	10 0

Key:

Code:	Instruction code (here 0006_{hex})
Bit number:	Enter here the number of the desired bit (0_{hex} to F_{hex} corresponds to 0_{dec} to 15_{dec}).
Res.:	This bit is reserved for future extensions. Set it to 0.
Output address:	Address of the word with the bit which was manipulated by the bit manipulation
Input address:	Address of the word with the bit determining the bit manipulation

Reset2_Out_In01 Instruction (0007_{hex})

Task:

This instruction causes a particular bit in the specified output word to be reset when the **positive** edge of a particular bit in the specified input word is detected.

Transfer the instruction to the controller board using the command *Receive_Processing_Instructions_Request* command ($005D_{hex}$).

Syntax:

Word 1	Code	
Word 2	Output address	
Word 3	000_{hex}	Bit number
Word 4	Input address	
Word 5	000_{hex}	Bit number

Bit	15 4	3 0

Key:

Code:	Instruction code (here 0007_{hex})
Bit number:	Enter here the number of the desired bit (0_{hex} to F_{hex} corresponds to 0_{dec} to 15_{dec}).
Output address:	Address of the word with the bit manipulated by the bit manipulation
Input address:	Address of the word with the bit determining the bit manipulation

Reset2_Out_In10 (0008_{hex}) Instruction

Task:

This instruction causes a particular bit in the specified output word to be reset when the **negative** edge of a particular bit in the specified input word is detected.

Transfer the instruction to the controller board using the command *Receive_Processing_Instructions_Request* command ($005D_{hex}$).

Syntax:

Word 1	Code	
Word 2	Output address	
Word 3	000_{hex}	Bit number
Word 4	Input address	
Word 5	000_{hex}	Bit number

Bit	15 4	3 0

Key:

Code:	Instruction code (here 0008_{hex})
Bit number:	Enter here the number of the desired bit (0_{hex} to F_{hex} corresponds to 0_{dec} to 15_{dec}).
Output address:	Address of the word with the bit manipulated by the bit manipulation
Input address:	Address of the word with the bit determining the bit manipulation

The instructions *Reset2_Out_In01* and *Reset2_Out_In10* are functionally equivalent with the instructions *Reset_Out_In01* and *Reset_Out_In10*. However, the instructions have a length of 5 words so that 16 bits are available for the addresses.

Notes on the addresses for conditional process data linkage

When the specified bit in the bit manipulation register has been set, the instructions behave like *Reset...01* and *Reset...10*. When the specified bit in the bit manipulation register has not been set, the instruction has no effect and no process data linkage takes place.

The bit manipulation register must be defined before the instructions *Reset_Out_In01_Switch* and *Reset_Out_In10_Switch* are used. For this purpose, enter a "pseudo module" (ID code 0013_{hex}) as the last module of the configuration. Then assign with logical addressing the address of the bit manipulation register to this pseudo module.

If you use the process input word of a real digital input module instead of the bit manipulation register, you can determine from within the system (e.g. with a keyswitch) whether the process data linkage is to be effective or not.

Reset2_Out_In01_Switch (0009$_{hex}$) Instruction

Task:

This instruction causes a particular bit in the specified output word to be reset when the **positive** edge of a particular bit in the specified input word is detected, provided that a particular bit in the bit manipulation register is set.

Transfer the instruction to the controller board using the command *Receive_Processing_Instructions_Request* (005D$_{hex}$).

Syntax:

Word 1	Code	
Word 2	BitmanRegAdr	
Word 3	000$_{hex}$	Bit number
Word 4	Output address	
Word 5	000$_{hex}$	Bit number
Word 6	Input address	
Word 7	000$_{hex}$	Bit number

Bit	15 4	3 0

Key:

Code:	Instruction code (here 0009$_{hex}$)
Bit number:	Enter here the number of the desired bit (0$_{hex}$ to F$_{hex}$ corresponds to 0$_{dec}$ to 15$_{dec}$).
BitmanRegAdr:	Address of the bit manipulation register
Output address:	Address of the word with the bit manipulated by the bit manipulation
Input address:	Address of the word with the bit determining the bit manipulation

Reset2_Out_In10_Switch (000A$_{hex}$) Instruction

Task:

This instruction causes a particular bit in the specified output word to be reset when the **negative** edge of a particular bit in the specified input word is detected, provided that a particular bit in the bit manipulation register is set.

Transfer the instruction to the controller board using the command *Receive_Processing_Instructions_Request* (005D$_{hex}$).

Syntax:

Word 1	Code	
Word 2	BitmanRegAdr	
Word 3	000$_{hex}$	Bit number
Word 4	Output address	
Word 5	000$_{hex}$	Bit number
Word 6	Input address	
Word 7	000$_{hex}$	Bit number

Bit	15 4	3 0

Key:

Code:	Instruction code (here 000A$_{hex}$)
Bit number:	Enter here the number of the desired bit (0$_{hex}$ to F$_{hex}$ corresponds to 0$_{dec}$ to 15$_{dec}$).
BitmanRegAdr:	Address of the bit manipulation register
Output address:	Address of the word with the bit manipulated by the bit manipulation
Input address:	Address of the word with the bit determining the bit manipulation

Bit_Copy Instruction ($000D_{hex}$)

Task: This instruction causes a particular bit in the specified input word to be copied to a particular bit in the specified output word.

Transfer the command to the controller board using the command *Receive_Processing_Instructions_Request* ($005D_{hex}$).

Syntax:

Word 1	Code	
Word 2	Output address	
Word 3	000_{hex}	Bit number
Word 4	Input address	
Word 5	000_{hex}	Bit number

Bit	15 4	3 0

Key:

Code:	Instruction code (here $000D_{hex}$)
Bit number:	Enter here the number of the desired bit (0_{hex} to F_{hex} corresponds to 0_{dec} to 15_{dec}).
Output address:	Address of the word to which the bit is copied
Input address:	Address of the word from which the bit is copied

Word_Copy_Bit_Mask Instruction ($000E_{hex}$)

Task: This instruction causes the bit-by-bit ANDing of a particular input word with a masking word and writes the result to a particular output word.

Transfer the command to the controller board using the command *Receive_Processing_Instructions_Request* ($005D_{hex}$).

Syntax:

Word 1	Code
Word 2	Output address
Word 4	Input address
	Mask

Bit	15 0

Key:

Code:	Instruction code (here $000E_{hex}$)
Output address:	Address of the output word to which the ANDing result is copied
Input address:	Address of the input word to be ANDed
Mask:	Masking word

8.9 Event Processing Commands

Some applications require specific reactions to particular events. With the event definition it is possible to be explicitly informed of signal transitions of particular inputs for this purpose.

Preparatory steps for event processing:

1. Define the desired events with the *Receive_Events_Request* command ($002A_{hex}$).
2. Define the message type with the *Set_Event_Message_Type_Request* command ($004D_{hex}$). The default setting is the *Event_Number_Indication* (8099_{hex})
3. Enable the desired events (e.g. with the *Enable_Event_Number_Request* command ($002B_{hex}$)).

The occurrence of all defined and enabled events is counted in specific counters. These event counters are reset with the enabling of the event. They can be read out with the *Read_Event_Counter_Request* command (0114_{hex}).

In connection with event processing please use logical addressing as far as possible. Event processing can on principle also be done under physical addressing, but this would mean that if you want to expand your system later on you would have to adapt all address specifications in all assignment lists. Under logical addressing you only need to add the new IBS devices to the assignment lists.

Receive_Events_Request$002A_{hex}$

Task: The parameters of the command allow the definition of up to 16 events. Every event definition occupies 5 words.

Syntax:

Word 1	Code				
Word 2	Parameter count				
Word 3	1st event ID				
Word 4	Event code				
Word 5	0_{hex}	00_{bin}	Address		1st event definition
Word 6	Mask 1				
Word 7	Mask 2				
	...				
	...				
	xth event ID				
	Event code				
	0_{hex}	00_{bin}	Address		xth event definition
	Mask 1				
Word (5z+2)	Mask 2				
Bit	15 12	11 10	9 0		

Key:	Code:	Command code (here $002A_{hex}$)
	Parameter count:	Number of subsequent words (here number (z) of the event definitions multiplied by 5).
	x:	Number of event definitions (1 to 16)
	Event ID:	It identifies the individual events. Assign for the 16 possible event definitions only event IDs that consist of a word with one bit set and 15 bits not set. Thus, the *Event_Number_Indication* message (8099_{hex}) can indicate all events in one word by bit-by-bit ORing of the event IDs (see the message description).
	Event code:	Assign the event code 0004_{hex} for the recognition of bit-state transitions with digital modules.
	Adresse:	Depending on the addressing mode used (see Section 6), enter here the logical or physical address of the input word for which you want to define an event. Use even byte addresses such as when assigning the logical addresses with the commands *Receive_Logical_IN_Address_Map_Req.* ($003A_{hex}$) *Receive_Logical_OUT_Address_Map_Req.* ($003B_{hex}$) (see Page 6-19).
	Mask 1:	Specifies within the data word determined by the address those bits for which the indication of an event is initiated by a positive edge. Set the desired bits to 1.
	Mask 2:	Specifies within the data word determined by the address those bits for which the indication of an event is initiated by a negative edge. Set the desired bits to 1.
Positive acknowledgment:	Quit_Receive_Event_Confirmation ($00BF_{hex}$)	
	Meaning:	The event definitions were accepted, but are not yet effective. Event definitions that had been defined with this command before were overwritten.
Negative acknowledgment:	Event_Error_Confirmation ($004C_{hex}$)	
	Meaning:	Event definition error. There are no valid event definitions; old definitions have been deleted.

Enable_All_Events_Request$002D_{hex}$

Task:	The command enables all previously defined event definitions.
Prerequisite:	The event definitions must be available and valid *(Receive_Events_Request* command ($002A_{hex}$)).
Syntax:	The command consists only of a single word, the command code ($002D_{hex}$). No further parameters follow.
Positive acknowledgment:	Quit_Enable_All_Events_Confirmation ($00DC_{hex}$)
Meaning:	All event definitions were enabled.

Negative acknowledgment: Event_Error_Confirmation ($004C_{hex}$)
Meaning: Event definition error. There are invalid event definitions.

Disable_All_Events_Request$002E_{hex}$

Task: The command disables all previously defined event definitions.

Prerequisite: The event definitions must exist and must be valid (*Receive_Events_Request* ($002A_{hex}$) command).

Positive acknowledgment: Quit_Disable_All_Events_Confirmation ($00DC_{hex}$)
Meaning: All event definitions were disabled.

Negative acknowledgment: No negative acknowledgment!

Syntax: The command consists only of one word, the command code ($002E_{hex}$). No further parameters follow.

Enable_Event_Number_Request$002B_{hex}$

Task: The command enables a particular event definition.

Prerequisite: The event definition must be available and must be valid (Receive_Events_Request command ($002A_{hex}$)).

Syntax:

Word 1	Code
Word 2	Parameter count
Word 3	Event ID

Bit	15 0

Key:
Code: Command code (here $002B_{hex}$)
Parameter count: Number of subsequent words (here 1).
Event ID: Enter here the ID of the event definition which you want to enable.

Positive acknowledgment: Quit_Enable_Event_Number_Confirmation ($00DA_{hex}$)
Meaning: The event definition has been enabled.

Negative acknowledgment: Event_Error_Confirmation ($004C_{hex}$)
Meaning: The specified event definition does not become effective.
- The specified event definition does not exist.
- The specified event definition is in the area of a disabled group.

Disable_Event_Number_Request002C$_{hex}$

Task: The command enables a particular event definition.

Prerequisite: The event definition must be available and valid (*Receive_Events_Request* (002A$_{hex}$) command).

Syntax:

Word 1	Code
Word 2	Parameter count
Word 3	Event ID

Bit	15 0

Key:

Code:	Command code (here 002C$_{hex}$)
Parameter count:	Number of subsequent words (here 1).
Event ID:	Enter here the ID of the event definition which you want to disable.

Positive acknowledgment: Quit_Disable_Event_Number_Confirmation (00DA$_{hex}$)
Meaning: The event definition has been disabled.

Negative acknowledgment: Event_Error_Confirmation (004C$_{hex}$)
Meaning: The specified event definition remains in the previous state.
- The specified event definition is in the area of a disabled group.

Enable_Event_Logical_Address_Request0036$_{hex}$

Task: The command enables the event definition for the input module specified via the logical address.

Prerequisite: The event definitions must be available and must be valid (*Receive_Events_Request* command (002A$_{hex}$)).

Syntax:

Word 1	Code		
Word 2	Parameter count		
Word 3	0$_{hex}$	00$_{bin}$	Input address

Bit	15 12	11 10	9 0

Key:

Code:	Command code (here 0036$_{hex}$)
Parameter count:	Number of subsequent words (here 1).
Input address:	Enter here, according to the addressing mode used, the logical or physical address (10 bits) of the input word, for which you want to enable the event definition. Use even

byte addresses such as those used when the logical addresses are assigned with the commands *Receive_Logical_IN_Address_Map_Req.* ($003A_{hex}$) *Receive_Logical_OUT_Address_Map_Req.* ($003B_{hex}$) (see Page 6-19).

Positive acknowledgment: Quit_Enable_Event_Logical_Address_Confirmation ($00E0_{hex}$)
Meaning: The selected event definition has been enabled.

Negative acknowledgment: Event_Error_Confirmation ($004C_{hex}$)
Meaning: The selected event definition does not become effective.
- The specified input address does not exist.
- There is no event definition for the specified input address.
- The specified input address is in the area of a disabled group.

Disable_Event_Logical_Address_Request0037_{hex}

Task: The command disables the event definition for the input module specified via the logical address.

Prerequisite: The event definition must be available and valid (*Receive_Events_Request* ($002A_{hex}$) command).

Syntax:

Word 1	Code		
Word 2	Parameter count		
Word 3	0_{hex}	00_{bin}	Input address

Bit	15 12	11 10	9 0

Key:
Code: Command code (here 0037_{hex})
Parameter count: Number of subsequent words (here 1).
Logical address: According to the addressing mode used, enter here the addressing mode used, the logical or physical address (10 bits) of the input word, for which you want to disable the event definition. Use even byte addresses such as used when the logical addresses are assigned with the commands *Receive_Logical_IN_Address_Map_Req.* ($003A_{hex}$) *Receive_Logical_OUT_Address_Map_Req.* ($003B_{hex}$) (see Page 6-19).

Positive acknowledgment: Quit_Enable_Event_Logical_Address_Confirmation ($00E1_{hex}$)
Meaning: The selected event definition has been disabled.

Negative acknowledgment: Event_Error_Confirmation ($004C_{hex}$)

Meaning: The selected event definition remains in the previous state.
- The specified input address does not exist.
- The specified input address is in the area of a disabled group.
- There is no event definition for the specified input address.

Set_Event_Message_Type_Request004D$_{hex}$

Task: This command defines the message type initiated after the occurrence of an event. Generated message types:

1 Event_Number_Indication (8099$_{hex}$)
This setting causes all occurred events to be transferred in one word. This word is generated by ORing the event IDs of all occurred events.

2 Event_Data_Indication (809B$_{hex}$)
This setting causes the event ID and the data word of the input module where the event was initiated to be transferred for each event occurred.

The default setting after the controller board startup is the message type *Event_Number_Indication* (8099$_{hex}$). Use the message type *Event_Data_Indication* (809B$_{hex}$) if the data at the moment of the event is to be used for further processing. This establishes a relationship between the data item and the moment of the event.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	000$_{hex}$	Message type

Bit	15 4	3 0

Key:

Code:	Command code (here 004D$_{hex}$)
Parameter count:	Number of subsequent words (here 1).
Message type:	Enter here the desired message type: 1: Event_Number_Indication (8099$_{hex}$) 2: Event_Data_Indication (809B$_{hex}$)

Positive acknowledgment: Quit_Set_Event_Message_Type_Confirmation (00DE$_{hex}$)

Negative acknowledgment: Set_Event_Message_Type_Failed_Confirmation (00DF$_{hex}$)

Read_Event_Counter_Request0114$_{hex}$

Task: There is an event counter for each of the up to 16 events. The *Read_Event_Counter_Request* (0114_{hex}) reads out the event counters. An event counter is incremented when the event with its ID is recognized.

Prerequisite: The events must have been defined and enabled. An event counter is set to 0 with the enabling (commands $002C_{hex}$, $002E_{hex}$ or 0036_{hex}) of the associated event.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	Event ID	for 1st event
Word 4	Event ID	for 2nd event
	...	
Word n+2	Event ID	for nth event

Bit	15 0

Key:

Code: Command code (here 0114_{hex})

Parameter count: Number of subsequent words (here number of event counters to be read out, max. 16_{dec}).

Event ID: Enter here the IDs of the event definitions whose counter you want to read out. The order of outputs in the acknowledgment corresponds to the order of your entries. If you enter a 0 as the only parameter, the counts of all events will be output. The order of outputs of the counts in the acknowledgment corresponds to the order specified during the event definition with the *Receive_Events_Request* ($0002A_{hex}$) command.

Acknowledgment: Quit_Read_Event_Counter_Confirmation ($811D_{hex}$)

Section 9

Messages of the IBS Master Board

This section provides information on

- the meanings and causes of the IBS master board messages;
- the parameters of the messages.

9 Messages of the IBS Master Board

Table 9-1: Messages of the IBS master board

Code	Message	Page
0007_{hex}	No_Command_Code_Indication	9-15
0008_{hex}	Command_Write_Error_Indication	9-15
0009_{hex}	Parameter_Write_Error_Indication	9-15
$000A_{hex}$	No_Command_Parameter_Routine_Indication	9-15
$000B_{hex}$	No_Error_Code_Indication	9-16
$000C_{hex}$	Controller_Parameter_Timeout_Indication	9-16
$000D_{hex}$	Host_Parameter_Timeout_Indication	9-16
0023_{hex}	CPU_Bus_Error_Indication	9-17
$002B_{hex}$	Logical_Address_Error_Confirmation	9-11
0038_{hex}	Bus_System_Error_Indication	9-17
$004A_{hex}$	No_Executable_Configuration_Confirmation	9-17
$004B_{hex}$	Command_Parameter_Error_Confirmation	9-17
$004C_{hex}$	Event_Error_Confirmation	9-34
$004D_{hex}$	IPMS_No_Error_Indication	9-18
$004E_{hex}$	Communication_Invalid_Indication	9-18
0059_{hex}	Unexpected_Group_Number_Confirmation	9-6
$005A_{hex}$	BK_Alarm_Failed_Confirmation	9-18
$005B_{hex}$	Unknown_Bus_Module_Confirmation	9-9
0066_{hex}	Receive_CR_Error_Confirmation	9-19
0068_{hex}	Check_Configuration_Error_Confirmation	9-8
0069_{hex}	Communication_Not_Ready_Confirmation	9-19
0088_{hex}	Start_Bus_Confirmation	9-14
$009C_{hex}$	Command_Disabled_Confirmation	9-31
$00AB_{hex}$	Physical_Configuration_Map_Valid_Confirmation	9-9
$00BD_{hex}$	Quit_Receive_Group_Numbers_Confirmation	9-6
$00BF_{hex}$	Quit_Receive_Events_Confirmation	9-34
$00C6_{hex}$	Stop_Bus_Confirmation	9-14
$00CA_{hex}$	Quit_Configure_Bus_Confirmation	9-10
$00CF_{hex}$	Quit_Set_Parameter_Timeout_Confirmation	9-31
$00D0_{hex}$	Quit_Receive_Logical_Localbus_Addressmap_Confirmation	9-11
$00D1_{hex}$	Quit_Receive_Logical_In_Addressmap_Confirmation	9-11
$00D2_{hex}$	Quit_Receive_Logical_Out_Addressmap_Confirmation	9-11
$00D3_{hex}$	Quit_Implement_Confirmation	9-11
$00D6_{hex}$	Quit_Enable_All_Messages_Confirmation	9-31
$00D7_{hex}$	Quit_Disable_All_Messages_Confirmation	9-31

Table 9-1: Messages of the IBS master board

Code	Message	Page
$00D8_{hex}$	Quit_Alarm_Stop_Confirmation	9-14
$00DA_{hex}$	Quit_Enable_Event_Number_Confirmation	9-34
$00DB_{hex}$	Quit_Disable_Event_Number_Confirmation	9-34
$00DC_{hex}$	Quit_Enable_All_Events_Confirmation	9-34
$00DD_{hex}$	Quit_Disable_All_Events_Confirmation	9-34
$00DE_{hex}$	Quit_Set_Event_Message_Type_Confirmation	9-34
$00DF_{hex}$	Set_Event_Message_Type_Failed_Confirmation	9-35
$00E0_{hex}$	Quit_Enable_Event_Logical_Address_Confirmation	9-35
$00E1_{hex}$	Quit_Disable_Event_Logical_Address_Confirmation	9-35
$00E2_{hex}$	Quit_Clear_Display_Confirmation	9-15
$00EC_{hex}$	Quit_Bus_Delay_Confirmation	9-15
$00ED_{hex}$	No_Map_Entry_Confirmation	9-19
$00F2_{hex}$	Quit_Receive_Instructions_Confirmation	9-33
$00F6_{hex}$	Quit_Groups_Error_Characteristic_Confirmation	9-8
$00FE_{hex}$	Quit_Module_Error_Ok_Confirmation	9-19
0105_{hex}	Quit_Receive_Localbus_Code_Map_Confirmation	9-13
8087_{hex}	Software_Revision_Confirmation	9-32
$809D_{hex}$	Switch_Group_Off_Confirmation	9-6
$809E_{hex}$	Switch_Group_On_Confirmation	9-7
$80A0_{hex}$	Module_Error_Indication	9-20
$80A1_{hex}$	Power_Fail_Indication	9-20
$80A2_{hex}$	Battery_Indication	9-21
$80C4_{hex}$ EE01 EE02 EE03 EE04 EE05 EE06	Bus_Error_Information_Indication Error type explanation and remedy instructions Error type explanation and remedy instructions Error type explanation and remedy instructions Error type explanation and remedy instructions Error type explanation and remedy instructions Error type explanation and remedy instructions	9-22 9-23 9-24 9-26 9-28 9-28 9-29
$80C5_{hex}$	Switch_Group_On_Failed_Confirmation	9-7
$80EE_{hex}$	Localbus_Module_Error_Confirmation	9-30
$80EF_{hex}$	Send_All_Module_Error_Confirmation	9-29
$80F3_{hex}$	Processing_Instructions_Error_Confirmation	9-33
$80F4_{hex}$	Send_Physical_Configuration_Confirmation	9-9
$80F5_{hex}$	Send_Logical_Address_Error_Confirmation	9-12
$80F7_{hex}$	Groups_Error_Characteristic_Failed_Confirmation	9-8
$80FF_{hex}$	Quit_Module_Error_Not_Possible_Confirmation	9-21
8119_{hex}	Send_Actual_Configuration_Confirmation	9-10
$811D_{hex}$	Quit_Read_Event_Counter_Confirmation	9-37

9.1 Format of a Message Description

This chapter describes the messages of the IBS master board. The messages are word-based; the message codes are given in hexadecimal notation. The description format is as follows.

Name of the messageMessage code$_{hex}$

Meaning: *Describes the message contents.*

Cause: *Describes the causes of the message.*

Remedy: *Gives, e.g. in the case of error messages, instructions on how to remove the cause of the error.*

Syntax: Only the message code is specified for a message without parameters.

The syntax of a message with parameters is given as a parameter block as follows:

Word 1	*Messasge code*
Word 2	*Number of subsequent words (parameter count)*
Word 3	*Parameter 1*
Word 4	*Parameter 2*
Word 5	*Parameter 3*
	...
Word n+2	*Parameter n*

Bit	15 0

Key: Parameter *Description of the individual parameters*

9.2 Configuration Messages

Receive_Group_Number_Failed_Confirmation0055$_{hex}$

Cause: The list transferred with the *Receive_Group_Numbers_Request* (0049$_{hex}$) contains faulty entries. Error causes can be:
- Invalid bus segment numbers.
- Multiple assignment of bus segment numbers.
- Invalid group number.
- An attempt was made to combine IBS devices without branching-off lines (no local bus connection, no remote bus branch) with other IBS devices in a group. Assign a separate group number to IBS devices without a branching-off line.
- Wrong number of parameters.

Remedy: Check the definition lists of the group definition.

Syntax: The message consists only of one word, the message code (0055$_{hex}$). No further parameters follow.

Unexpected_Group_Number_Confirmation0059$_{hex}$

Meaning: Error when executing the commands *Switch_Group_Off_Request* (0021$_{hex}$) or *Switch_Group_On_Request* (0020$_{hex}$)
- It was attempted to disable a device that cannot be disabled.
- On enabling or disabling of a group, a group number was specified which had not been defined before.
- The transferred group number is outside the permissible range.

The error is only indicated, without affecting the behavior of the controller board.

Remedy: Check the application program for non-existent group numbers and wrong numbers of parameters.

Syntax: The message consists only of one word, the message code (0059$_{hex}$). No further parameters follow.

Quit_Receive_Group_Numbers_Confirmation00BD$_{hex}$

Meaning: The *Receive_Group_Numbers_Request* command (0049$_{hex}$) was executed successfully.

Syntax: The message consists only of one word, the message code (00BD$_{hex}$). No further parameters follow.

Switch_Group_Off_Confirmation809D$_{hex}$

Meaning: A group of bus segments was disabled. As parameter, the number of the disabled group is transferred.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Group number

Bit	15 8	7 0

Key:

Code:	Message code (here $809D_{hex}$)
Parameter count:	Number of subsequent words (here 1)
Group number:	Number of the group that was disabled. (00_{hex} to FF_{hex} corresponds to 0 to 255_{dec})

Switch_Group_On_Confirmation$809E_{hex}$

Meaning:

A group of bus segments was enabled. As parameter, the number of the enabled group is transferred.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Group number

Bit	15 8	7 0

Key:

Code:	Message code (here $809E_{hex}$)
Parameter count:	Number of subsequent words (here 1)
Group number:	Number of the group that was enabled. (00_{hex} to FF_{hex} corresponds to 0 to 255_{dec})

Switch_Group_On_Failed_Confirmation$80C5_{hex}$

Meaning:

The attempt to enable a group was not successful. As parameter, the group number is transferred.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Group number

Bit	15 8	7 0

Key:

Code:	Message code (here $80C5_{hex}$)
Parameter count:	Number of subsequent words (here 1)
Group number:	Number of the group that could not be enabled (00_{hex} to FF_{hex} corresponds to 0 to 255_{dec}).

Quit_Groups_Error_Characteristic_Confirmation00F6$_{hex}$

Meaning: The *Define_Groups_Error_Characteristic_Request* (0060$_{hex}$) command was executed.

Syntax: The message consists only of one word, the message code (00F6$_{hex}$). No further parameters follow.

Groups_Error_Characteristic_Failed_Confirmation80F7$_{hex}$

Meaning: The *Define_Groups_Error_Characteristic_Request* (0060$_{hex}$) command was not executed without errors. The error number identifies the error type. The information contained in it has not been taken over. The parameter number identifies the faulty parameter.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00$_{hex}$	Error number
Word 4	00$_{hex}$	Parameter number

Bit	15 8	7 0

Key:

Code: Message code (here 80F7$_{hex}$)
Parameter count: Number of subsequent words (here 2)
Error number: The *error number* parameter identifies the type of the error:
1 Elements exist more than once.
2 The group number does not exist.
Parameter number: The *parameter number* parameter identifies the faulty parameter in the command *Define_Groups_Error_Characteristic_Request* (0060$_{hex}$).

Check_Configuration_Error_Confirmation0068$_{hex}$

Meaning: This error is indicated when, with the *Check_Physical_Configuration_Map_Request* (0058$_{hex}$) command the configuration and transferred to the controller board and the currently connected bus configuration are not identical.

Remedy: Check the ID code list for
- the number of parameters,
- identical orders of ID codes and modules, and
- the positions of the registers and special ID codes.

Syntax: The message consists only of one word, the message code (0068$_{hex}$). No further parameters follow.

Unknown_Bus_Module_Confirmation005B$_{hex}$

Meaning: An invalid length code was specified in the *Check_Physical_Configuration* (0058_{hex}) command;
an invalid number of process data items was specified;
the first module is no module with BK module functionality.

Remedy: Check the parameterization list of the ID codes for invalid ID codes and non-existent length codes and the number of process data.

The bus goes without *Reset* into the STOP state. The output data is not reset. The bus data cycle can only be reinitialized by the *Warmstart_Request* command ($004C_{hex}$) or the reset button.

Syntax: The message consists only of one word, the message code ($005B_{hex}$). No further parameters follow.

Physical_Configuration_Map_Valid_Confirmation00AB$_{hex}$

Meaning: The bus configuration transferred to the controller board with the *Receive_Physical_Configuration_Map_Request* command is not identical with the currently connected configuration.

Syntax: The message consists only of one word, the message code ($00AB_{hex}$). No further parameters follow.

Send_Physical_Configuration_Confirmation80F4$_{hex}$

Meaning: Positive acknowledgment of the *Send_Physical_Configuration_Request* command ($005E_{hex}$). It contains, in the form of length and ID codes, the physical bus configuration stored in the controller board RAM.

Syntax:

Word 1	Code		
Word 2	Parameter count (n)		
Word 3	Length code	ID code	for device 1
Word 4	Length code	ID code	for device 2
Word 5	Length code	ID code	for device 3
	...	...	
Word n+2	Length code	ID code	for device n

Bit	15 8	7 0

Key:

Code:	Message code (here $80F4_{hex}$)
Parameter count:	Number of subsequent words (here: number of devices).
Length code:	The length code describes the address space requirements of the IBS device in the host.
ID code:	ID code of the IBS device. It is printed on the modules in decimal form as a *Module ID* . (0_{dec} to 255_{dec} corresponds to 00_{hex} to FF_{hex})

The length code of IBS devices supporting communication (PCP) contains only their process data words.

Send_Actual_Configuration_Confirmation8119$_{hex}$

Meaning: Positive acknowledgment of the *Send_Actual_Configuration_Request* ($010D_{hex}$) command. It contains the currently connected bus configuration in the form of length and ID codes.

Syntax:

Word 1	Code		
Word 2	Parameter count (n)		
Word 3	Result	00_{hex}	
Word 4	Length code	ID code	for device no. 1
Word 5	Length code	ID code	for device no. 2
Word 6	Length code	ID code	for device no. 3
	...	...	
Word n+2	Length code	ID code	for device no. n
Bit	15 8	7 0	

Key:

Code: Message code (here 8119_{hex})

Parameter count: Number of subsequent words (here: number of devices).

Result: The *Result* parameter is always 00_{hex}. When the controller board can read in only part of the currently connected bus configuration (as, for example, the remote bus cable is defective at some point), the message contains only the length and ID codes of the accessible IBS devices. If the controller board could not read in the connected bus configuration at all (if, for example, the remote bus cable is not connected to the controller board), this message is also output, but without length and ID code.

Length code: The length code describes the address space requirements of the IBS device in the host.

ID code: ID code of the IBS devices. It is printed on the modules in decimal form as a *Module ID*.
(0 to 255_{dec} corresponds to 00_{hex} to FF_{hex})

The length of IBS devices supporting communication (PCP) contains only their process data words.

Quit_Configure_Bus_Confirmation00CA$_{hex}$

Meaning: The *Configure_Bus_Request* (0023_{hex}) command was executed.

Syntax: The message consists only of one word, the message code ($00CA_{hex}$). No further parameters follow.

9.3 Addressing Messages

Logical_Address_Error_Confirmation002B$_{hex}$

Meaning: A check revealed an error in the logical addressing lists sent to the controller board.

The bus goes without reset into the STOP state. The outputs are not reset.

In addition, the *Send_Log_Address_Error_Request* command (005F$_{hex}$) provides error diagnostics.

Syntax: The message consists only of one word, the message code (002B$_{hex}$). No further parameters follow.

Quit_Receive_Logical_Localbus_Addressmap_Confirmation00D0$_{hex}$

Meaning: The *Receive_Logical_Localbus_Addressmap_Request* command (0039$_{hex}$) was executed.

Syntax: The message consists only of one word, the message code (00D0$_{hex}$). No further parameters follow.

Quit_Receive_Logical_In_Addressmap_Confirmation00D1$_{hex}$

Meaning: The *Receive_Logical_In_Addressmap_Request* command (003A$_{hex}$) was executed.

Syntax: The message consists only of one word, the message code (00D1$_{hex}$). No further parameters follow.

Quit_Receive_Logical_Out_Addressmap_Confirmation00D2$_{hex}$

Meaning: The *Receive_Logical_Out_Address_Map_Request* (003B$_{hex}$) command was executed.

Syntax: The message consists only of one word, the message code (00D2$_{hex}$). No further parameters follow.

Quit_Implement_Confirmation00D3$_{hex}$

Meaning: The Implement_All_Logical_Address_Maps_Request command (0040$_{hex}$) was executed. The logical addressing lists sent to the controller board contain no errors.

Syntax: The message consists only of one word, the message code (00D3$_{hex}$). No further parameters follow.

Send_Logical_Address_Error_Confirmation80F5$_{hex}$

Meaning:

This message is the positive acknowledgment of the command *Send_Log_Address_Error_Request* (005F$_{hex}$) and communicates the last logical addressing error. If no error was found, the *list*, *parameter number*, and *error number* parameter values are 0.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00$_{hex}$	List
Word 4	00$_{hex}$	Parameter number
	00$_{hex}$	Error number

Bit	15 8	7 0

Key:

Code: Message code (here 80F5$_{hex}$)

Parameter count: Number of subsequent words

List: The *list* parameter identifies the faulty list:
- 0 No error found
- 2 Logical IN address list (*Receive_Logical_IN_Address_Map_Request* command (003A$_{hex}$))
- 3 Logical OUT address list (*Receive_Logical_OUT_Address_Map_Request* (003B$_{hex}$) command)
- 4 ID list *Check_Physical_Configuration_Request* (0058$_{hex}$) command)
- 6 Logical-Bus-Code-Map *Receive_Localbus_Code_Map_Request* command (0069$_{hex}$)

Parameter number: The *parameter number* parameter refers to the faulty parameter in this list:
- If the error number is 1, *parameter number* specifies the number of the first missing parameter in the list of logical addresses.
- If the error number is 2, *parameter number* refers to the number of the first parameter specified too many in the list of the logical addresses.
- Otherwise, the number of the faulty entry will be specified.

Error number: The *error number* identifies the type of the error:
- 0 No error detected
- 1 Too many entries in the list that was sent with the faulty command.
- 2 Too many entries in the list sent with the faulty command.
- 3 An invalid bus segment number was assigned. 0 to 255$_{dec}$ (corresponds to 00$_{hex}$ to FF$_{hex}$) are permissible.
- 4 Multiple assignment of a bus segment number.

5 Invalid host address area
7 The address area of two IBS devices overlap.
8 An odd address was assigned for an IBS device with more than 8 bits of process data. Assign an even address.
9 An invalid length code was used.
10 A sent ID code is not identical with the IBS device in the system.
12 The specified address is invalid.
14 An invalid remote bus level was specified. Firmware 3.x allows only *remote bus levels 0 and 1*.
15 The wrong remote bus level was specified for an IBS device. Specify the *remote bus level 0* for main-line IBS devices, and *remote bus level 1* for IBS devices in the remote bus branch.
16 No command execution possible, as the bus is in the Run state. In this case the *list* and *parameter number* parameter values are 0.
17 An invalid data consistency was specified.

Quit_Receive_Localbus_Code_Map_Confirmation0105$_{hex}$

Meaning: The *Receive_Localbus_Code_Map_Request* (0069$_{hex}$) command was executed successfully.

Syntax: The message consists only of one word, the message code (0105$_{hex}$). No further parameters follow.

9.4 Operation Messages

Start_Bus_Confirmation0088$_{hex}$

Meaning: The cyclic data traffic on the bus has been started. The controller board operates the process data traffic from this moment onwards. This means that a process image of the host input data is cyclically generated, and that the process output data from the host is transferred cyclically to the modules connected to the bus. In addition, the controller board operates the **P**eripherals **C**ommunication **P**rotocol (PCP) for modules working with PCP.

Syntax: The message consists only of one word, the message code (0088_{hex}). No further parameters follow.

Stop_Bus_Confirmation00C6$_{hex}$

Meaning: The *Stop_Bus_Cycle_Request* (0002_{hex}) command was executed. The bus is in the STOP state.

Process data: The cyclic data traffic on the bus has been stopped. The existing process data image of the modules connected to the bus is statically retained without being updated.

The command does not switch the process output data into the safe state (reset of the outputs). This is only carried out with the *Alarm_Stop_Request* command ($004A_{hex}$).

PCP channel: The operation of the **P**eripherals **C**ommunication **P**rotocol (PCP) does not continue. The established connections are **not** automatically aborted. The processing of any outstanding services will continue after the restart of the data traffic.

Syntax: The message consists only of one word, the message code ($00C6_{hex}$). No further parameters follow.

Quit_Alarm_Stop_Confirmation00D8$_{hex}$

Meaning: The *Alarm_Stop_Request* ($004A_{hex}$) was executed successfully. The bus is in the STOP state.

Syntax: The message consists only of one word, the message code ($00D8_{hex}$). No further parameters follow.

Start_Bus_Not_Possible_Confirmation00E3$_{hex}$

Meaning: This is a negative acknowledgment of the *Start_Bus_Cycle_Request* command (0001_{hex}). It is sent when no data traffic is possible on the bus.

Causes: Hardware error on the controller board;
Configuration change on the local bus (modules have been removed or added).

Syntax: The message consists only of one word, the message code ($00E3_{hex}$). No further parameters follow.

Quit_Bus_Delay_Confirmation00EC$_{hex}$

Meaning: The *Bus_Delay_Request* command (0059$_{hex}$) was executed successfully.

Syntax: The message consists only of one word, the message code (00EC$_{hex}$). No further parameters follow.

9.5 Error Handling Messages

Quit_Clear_Display_Confirmation00E2$_{hex}$

Meaning: The *Clear_Display_Request* (004E$_{hex}$) command was executed.

Syntax: The message consists only of one word, the message code (00E2$_{hex}$). No further parameters follow.

No_Command_Code_Indication0007$_{hex}$

Meaning: A non-defined command code was sent to the controller board.

Remedy: Check the application program.

Syntax: The message consists only of one word, the message code (0007$_{hex}$). No further parameters follow.

Command_Write_Error_Indication0008$_{hex}$

Meaning: A command with a wrong number of parameters was sent to the controller board.

Remedy: Check the application program (e.g. PCP commands)

Syntax: The message consists only of one word, the message code (0008$_{hex}$). No further parameters follow.

Parameter_Write_Error_Indication0009$_{hex}$

Meaning: A command with parameters was sent with the parameter count *0*.

Remedy: Check the parameters in the application program.

Syntax: The message consists only of one word, the message code (0009$_{hex}$). No further parameters follow.

No_Command_Parameter_Routine_Indication000A$_{hex}$

Meaning: An unknown command was sent.

Remedy: Check the appication program.

Syntax: The message consists only of one word, the message code (000A$_{hex}$). No further parameters follow.

No_Error_Code_Indication000B$_{hex}$

Meaning: An unknown error code has occurred.

Remedy: Please consult Phoenix Contact.

Syntax: The message consists only of one word, the message code (000B$_{hex}$). No further parameters follow.

Controller_Parameter_Timeout_Indication000C$_{hex}$

Meaning: The error messages is issued when:
- a pending message was not fetched within the first 8 minutes (*TIME-OUT*) and, therefore, blocks the MPM area involved;
- a pending message was not fetched and a second one has been pending for 8.1 seconds.

Remedy: Check whether the presence of a message is checked in your program at sufficiently short intervals.

The *Controller_Parameter_Timeout_Indication* message overwrites all messages which have occurred after the message that was not fetched from the MPM, are ready on the IBS master board for transfer to the MPM, but could not be transferred, as the first message that was not fetched is blocking the MPM area.

This message disables any further error messages. Bus diagnostics are only possible with the *GetIBSDiagnostics* function.

Syntax: The message consists only of one word, the message code (000C$_{hex}$). No further parameters follow.

Host_Parameter_Timeout_Indication000D$_{hex}$

Meaning: The controller board is unable to acknowledge an arrived command, as the host has not yet reset the handshake bit for the previous command acknowledgment. This handshake bit was sent by the host when the previous command was sent.

Remedy: Check the application program.

This message disables any further error messages. Bus diagnostics are only possible with the *GetIBSDiagnostics* function.

Syntax: The message consists only of one word, the message code (000D$_{hex}$). No further parameters follow.

CPU_Bus_Error_Indication0023$_{hex}$

Meaning: The watchdog has tripped.

Remedy: Please consult Phoenix Contact.

Syntax: The message consists only of one word, the message code (0023$_{hex}$). No further parameters follow.

Bus_System_Error_Indication0038$_{hex}$

Meaning: An error causing the system to be no longer operable with the last configuration has occurred. The controller board immediately indicates the error. Then a test routine locating the error in the system is automatically executed.

On termination of this test routine, a detalied error message is provided with the *Bus_Error_Information_Indication* message (80C4$_{hex}$)

The error is issued only as a message. No data cycle is run while the error is being located.

Syntax: The message consists only of one word, the message code (0038$_{hex}$). No further parameters follow.

No_Executable_Configuration_Confirmation004A$_{hex}$

Meaning: A non-existent configuration was sent to the controller board.

Remedy: Check the connection of the remote bus cable and carry out a reset on the controller board.

Syntax: The message consists only of one word, the message code (004A$_{hex}$). No further parameters follow.

Command_Parameter_Error_Confirmation004B$_{hex}$

Meaning: The number of parameters for the command directly preceding this message is too great or too small.

The command execution is aborted.

Remedy: Check in your program the command sent last for a wrong number of parameters.

Syntax: The message consists only of one word, the message code (004B$_{hex}$). No further parameters follow.

IPMS_No_Error_Indication004D$_{hex}$

Meaning: The IPMS protocol chip detected a bus error, but no bit has been set in the IPMS error register.

Remedy: Please consult Phoenix Contact.

Carry out a reset on the controller board.

Syntax: The message consists only of one word, the message code (004D$_{hex}$). No further parameters follow.

Communication_Invalid_Indication004E$_{hex}$

Meaning: The controller board uses a fixed memory area for the command processing. When these resources are exhausted, this error message is generated. This may be the case, for example, when a greater number of commands is issued within a short time and the controller board is unable to execute them fast enough.

Remedy: Check your program.

Syntax: The message consists only of one word, the message code (004E$_{hex}$). No further parameters follow.

BK_Alarm_Failed_Confirmation005A$_{hex}$

Meaning: One of the following commands was not executed successfully:
- Set-BK-Alarm-Logical-Request (0024$_{hex}$)
- Reset-BK-Alarm-Logical-Request (0025$_{hex}$)
- Set-BK-Alarm-Physical-Request (0026$_{hex}$)
- Reset-BK-Alarm-Physical-Request (0027$_{hex}$)

Possible error causes are:
a) You specified a non-existent bus segment number.
b) You specified an invalid number of parameters.

Remedy: Check your application program.

Syntax: The message consists only of one word, the message code (005A$_{hex}$). No further parameters follow.

Receive_CR_Error_Confirmation0066$_{hex}$

Meaning: This error message is sent in response to a check of the *Receive Communication Reference* command, when
- an invalid communication reference was used,
- too many parameters were sent, or
- not enough parameters were sent.

Remedy: Check the communication relationship list (CRL) for valid communication references (2 to 62). The assigned communication references must succeed one another without a gap (e.g. 2, 3, 4, 5,...). The call order is irrelevant (e.g. 4, 2, 5, 3 is also allowed).

Syntax: The message consists only of one word, the message code (0066$_{hex}$). No further parameters follow.

Communication_Not_Ready_Confirmation0069$_{hex}$

Meaning: A request or respond command was sent although the communication (short: PCP) has not been initialized.

Remedy: Initialize the communication in your application program.

See the manual for the fundamentals and the use of the Peripherals Communication Protocol, *IBS PCP UM E* (Order No. 27 53 93 1).

Syntax: The message consists only of one word, the message code (0069$_{hex}$). No further parameters follow.

No_Map_Entry_Confirmation00ED$_{hex}$

Meaning: This message is sent when there are no entries for commands which are to send a list or table in any form.

Remedy: Check your application program

Syntax: The message consists only of one word, the message code (00ED$_{hex}$). No further parameters follow.

Quit_Module_Error_Ok_Confirmation00FE$_{hex}$

Meaning: This message is the positive acknowledgment of the *Quit_Module_Error_Request* (0064$_{hex}$) and *Quit_Module_Error_All_Request* (0065$_{hex}$) commands.

Syntax: The message consists only of one word, the message code (00FE$_{hex}$). No further parameters follow.

Module_Error_Indication80A0$_{hex}$

Meaning:

An error was found on one or more IBS devices with error indication feature. Depending on the addressing type used, the parameters indicate the logical or the physical addresses of the affected bus segments.

Causes:

- I/O voltage failure
- Blown fuse
- The current limiting feature has responded

Syntax:

Word 1	Code	
Word 2	Parameter count (n)	
Word 3	00_{hex}	Bus segment
Word 4	00_{hex}	Bus segment
	...	
Word n+2	00_{hex}	Bus segment

Bit	15 0

Key:

Code:	Message code (here 80A0$_{hex}$)
Parameter count:	Number of subsequent words (here number of bus segments where a module error was detected)
Bus segment:	Bus segment number of the affected bus segment (00_{hex} to FF_{hex} corresponds to 0 to 255_{dec})

Power_Fail_Indication80A1$_{hex}$

Meaning:

The voltage supply (9 V) of a local bus has failed. Depending on the addressing type used, the parameters indicate the logical or the physical addresses of the affected bus segments.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Bus segment

Bit	15 0

Key:

Code:	Message code (here 80A1$_{hex}$)
Parameter count:	Number of subsequent words (here 1)
Bus segment:	Bus segment number of the affected bus segment (00_{hex} to FF_{hex} corresponds to 0 to 255_{dec})

Battery_Indication80A2$_{hex}$

Meaning:

On some special modules the logic voltage (U_L) is backed up by a battery. If this supply voltage fails, this message indicates switching over to battery operation. The parameter specifies the logical or physical bus segment number of the affected bus terminal module.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00_{hex}	Bus segment

Bit	15 0

Key:

Code:	Message code (here 80A2$_{hex}$)
Parameter count:	Number of subsequent words (here 1)
Bus segment:	Bus segment number of the affected bus segment (00_{hex} to FF_{hex} corresponds to 0 to 255_{dec})

Quit_Module_Error_Not_Possible_Confirmation80FF$_{hex}$

Meaning:

This message is the negative acknowledgement of the commands *Quit_Module_Error_Request* (0064_{hex}) and *Quit_Module_Error_All_Request* (0065_{hex}).

Syntax:

Word 1	Code
Word 2	Parameter count
Word 3	Parameter number
Word 4	Error number

Bit	15 0

Key:

Code:	Message code (here 80FF$_{hex}$)
Parameter count:	Number of subsequent words (here 2)
Parameter number:	The *parameter number* identifies the faulty parameter.
Error number:	The *error number* number specifies the type of the error: 1 Odd number of parameters (parameter number 1). 2 Unknown bus segment address. 3 The specified module (number) does not exist. 4 The specified module is currently not accessible, as its segment has been disabled. 5 The required ID cycles could not be executed successfully.

Only error number 5 is used in the response to the *Quit_Module_Error_All_Request* (0065_{hex}) command.

Bus_Error_Information_Indication80C4$_{hex}$

First comes the general format of the message. The format depends on the *error type* parameter, which is described in detail below.

Meaning: This message follows the *Bus_System_Error_Indication* (0038$_{hex}$) error message. It provides detailed information on the bus system's error state.

Consequence: The bus is in the STOP state and the outputs have been set to *0*.

Syntax:

Word 1	Code
Word 2	Parameter count (n)
Word 3	Error type
Word 4	Error description
	...
Word n+2	Error description

Bit	15 0

Key:

Code:	Message code (here 80C4$_{hex}$)
Parameter count:	Number of subsequent words
Error type:	EE01, EE02, EE03, EE04, EE05 or EE06. Depending on the error type, the other parameters classify the error as follow-up information.
Error description:	Optional follow-up information with the error types EE02 and EE03. The individual parameters are explained in the following descriptions of the individual error types.

Error type EE01

Meaning: A bus check did not reveal an error in the currently permissible configuration. However, there is an error in the bus installation.

Cause: A short disruption of the data transmission.

Remedy: Check the system for:
- missing or improper bus cable shielding (connectors),
- missing or improper grounding/potential equalization,
- voltage dips on the logic supply of the remote bus devices.

Syntax:

Word 1	Code
Word 2	Parameter count
Word 3	Error type

Bit	15 0

Key:

Code:	Message code (here $80C4_{hex}$)
Parameter count:	Number of subsequent words (here 1)
Error type:	Here EE01

Error type EE02

Meaning: Configuration changes which do not allow that the data traffic on the bus continues.

Cause:
- The maximum permissible number of IBS words was exceeded.
- The maximum permissible number of IBS modules was exceeded.

Syntax:

Word 1	Code
Word 2	Parameter count
Word 3	Error type
Word 4	Error number

Bit	15 0

Key:

Code:	Message code (here $80C4_{hex}$)
Parameter count:	Number of subsequent words (here 2)
Error type:	Here EE02
Error number:	The individual system errors are encoded as error numbers in word 4. The following table lists all possible error numbers with a short description of the error.

Table 9-2: Meaning of the *error number* parameter

Error number	Meaning
$DD01_{hex}$	An ID code stored in the controller board RAM does not match the associated bus terminal module in the system (e.g. due to module exchange or device error).
$DD02_{hex}$	An ID code stored in the controller board RAM does not match the associated I/O bus terminal module in the system (e.g. owing to module exchange or device error).
$DD03_{hex}$	A specified local bus does not exist in the system, or a local bus cable is defective.
$DD04_{hex}$	A local bus has more devices than expected, as, for example, a device was added while the bus was in operation.
$DD05_{hex}$	A local bus has fewer devices than expected, as, for example, a device was removed while the bus was in operation.
$DD06_{hex}$	An ID code stored in the controller board RAM does not match a local bus device in the system (e.g. owing to module exchange or device error)
$DD07_{hex}$	The configuration could not be read in, as the remote bus cable is not connected to the controller board.
$DD08_{hex}$	A local bus which is not stored in the controller board RAM is connected to a bus terminal module with I/O functionality (as, for example, local bus devices were added while the bus was in operation).

Table 9-2: Meaning of the *error number* parameter

Error number	Meaning
$DD09_{hex}$	The connected configuration is shorter than expected, as the remote bus was shortened compared with the configuration stored in the controller board RAM (remote bus device removed or remote bus cable not connected).
$DD0A_{hex}$	Multiple transmission errors between 2 error-free data cycles (corresponds to error type EE06). No error was found in the configuration during acquisition and comparison of the bus configuration.
$DD0B_{hex}$	The connected configuration is longer than expected, as the remote bus has been expanded compared with the configuration stored in the controller board RAM.
$DD0C_{hex}$	The maximum permissible configuration has been exceeded. Up to 256 bus terminal modules and 256 words (total of all register lengths and PCP words) are permissible on the data ring.
$DD11_{hex}$	- The data register of a remote bus device has been interrupted. - A length code stored in the controller board RAM does not match the associated remote bus device in the system (e.g. owing to module exchange or device error).
$DD12_{hex}$	- The data register of a local bus device has been interrupted. - A length code stored in the controller board RAM does not match the associated local bus device in the system (e.g. due to module exchange or device error).
$DD15_{hex}$	Short-time change of an ID code in operation with a specified bus terminal module or module in the specified local bus (similar to DD01/02/06)
$DD18_{hex}$	Short-time error in a local bus (8-wire technology) in operation, due to cable or module error (similar to DD03/05/08).
$DD19_{hex}$	A bus interruption or the voltage reset of a device was detected in the additional diagnostics phase.
$DD1A_{hex}$	Multiple transmission errors between 2 error-free data cycles (corresponds to error type EE06) detected in the additional diagnostics phase.
$DD2B_{hex}$	The connected configuration is longer than expected, as the remote bus has been expanded compared with the configuration stored in the controller board RAM. The error was detected in the additional diagnostics phase.
$DD42_{hex}$	Short-time bus interruption due to the voltage reset of a device, or defective bridge (RBST) in an outgoing remote bus connector.
$DD50_{hex}$	A remote bus or local bus error was detected in the additional diagnostics phase.
$DD51_{hex}$	A local bus error was detected in the additional diagnostics phase.
$DD52_{hex}$	A remote bus error was detected in the additional diagnostics phase.

Error type EE03

Meaning: An error due to an electrical modification of the bus configuration (e.g. open circuit) occurred while the bus was in operation.

Cause:
- Remote bus: Defective bus terminal module or upstream remote bus cable
- Local bus: Defective local bus cable or module

Consequences: The data cycle is disabled - unless the errors occurred only in the groups for which it was previously defined with the *Define_Groups_Error_Characteristic_Request* command (0060_{hex}) that the rest of the bus may be operated without them.

Syntax:

Word 1	Code	
Word 2	Parameter count (n)	
Word 3	Error type	
Word 4	$FF01_{hex}$	Parameter block *Groups*
Word 5	Number of groups	
Word 6	1st group number	
Word 7	...	
Word 8	xth group number	
	...	
	...	
	$FF02_{hex}$	Parameter block *Remote bus*
	Number of RBs	
	1st remote bus number	
	1st error number	
	...	
	xth remote bus number	
	xth error number	
	...	
	...	
	$FF03_{hex}$	Parameter block *Local bus*
	Number of local buses	
	1st local bus number	
	1st error number	
Word n	...	
Word n+1	xth local bus number	
Word n+2	xth error number	

Bit	15 0

Key:

Code:	Message code (here $80C4_{hex}$)
Parameter count:	Number of subsequent words
Error type :	Here EE03
FF01	Beginning of the parameter block indicating the faulty groups.
Number of groups:	Number of faulty groups.
1st group number:	Number of the 1st faulty group.
xth group number:	Number of the last faulty group.
FF02	Beginning of the parameter block indicating the faulty remote bus segments.
Number of RBs:	Number of faulty remote bus segments
1st rem. bus number:	Bus segment number of the 1st faulty remote bus
1st error number:	This error number describes the error of the remote bus specified in the previous line (see Table 9-2).
xth rem. bus number:	Bus segment number of the last faulty remote bus
xth error number:	This error number describes the error of the remote bus specified in the previous line (see Table 9-2).
FF03:	Beginning of the parameter block indicating the bus segments with an error in the local bus.
Number of local buses:	Number of bus segments with an error in the local bus.
1st local bus number:	Bus segment number of the 1st faulty local bus.
1st error number:	This error number describes the error of the local bus specified in the previous line (see Table 9-2).
xth local bus number:	Bus segment number of the last faulty local bus.
xth error number:	This error number describes the error of the local bus specified in the previous line (see Table 9-2).

The length of the *groups*, *remote bus* and *local bus* parameter blocks depends on the number of errors that have occurred. All entries are four-digit hexadecimal.

Error type EE04

Meaning: The bus configuration could not be read in.

Cause: The voltage supply for the electronics (logic voltage) of one or more remote bus device is disrupted.

Remedy: Check the system for voltage dips on the remote bus device supply line (logic voltage).

Syntax:

Word 1	Code
Word 2	Parameter count
Word 3	Error type

Bit	15 0

Key:

Code:	Message code (here $80C4_{hex}$)
Parameter count:	Number of subsequent words (here 1)
Error type :	here EE04

Error type EE05

Meaning: All InterBus-S groups are disabled.

Cause: The last group which was still in operation has been disabled.

Remedy: Check your application program.

Syntax:

Word 1	Code
Word 2	Parameter count
Word 3	Error type

Bit	15 0

Key:

Code:	Message code (here $80C4_{hex}$)
Parameter count:	Number of subsequent words (here 1)
Error type :	here EE05

Error type EE06

Meaning: No configuration error was found during the bus configuration acquisition and comparison. However, data cycles are not possible.

Cause:
- Defect of an IBS device
- Installation error (see also error type EE01)
- Occurrence of a multiple error which cannot be located
- Unidentifiable errors

Remedy: Check your bus configuration

Syntax:

Word 1	Code
Word 2	Parameter count
Word 3	Error type

Bit	15 0

Key:
- Code: Message code (here $80C4_{hex}$)
- Parameter count: Number of subsequent words (here 1)
- Error type : Here EE06

Send_All_Module_Error_Confirmation$80EF_{hex}$

Meaning: This message is initiated by the *Send_All_Module_Error_Request* command ($005C_{hex}$). The list contains all bus segments where a module indicates an error.

Syntax:

Word 1	Code
Word 2	Parameter count (n)
Word 3	1st bus segment
Word 4	2nd bus segment
	...
Word n+2	xth bus segment

Bit	15 0

Key:
- Code: Message code (here $80EF_{hex}$)
- Parameter count: Number of subsequent words (here number of bus segments where a module error was found)
- Bus segment: Bus segment number for identifying the error locations (00_{hex} to FF_{hex} corresponds to 0 to 255_{dec})

Localbus_Module_Error_Confirmation80EE$_{hex}$

Meaning:

This message is initiated by the *Send_Localbus_Module_Error_Request* (005B$_{hex}$) command. It contains a list of all error-indicating modules of the local bus specified in the *Send_Localbus_Module_Error_Request* (with position and ID code).

Syntax:

Word 1	Code	
Word 2	Parameter count (n)	
Word 3	1st position	for 1st error-indicating module
Word 4	1st ID code	
	...	
	xth position	für xth error-indicating module
Word n+2	xth ID code	

Bit	15 0

Key:

Code:	Message code (here 80EE$_{hex}$)
Parameter count:	Number of subsequent words (here number of bus segments where a module error was found, multiplied by 2)
Position:	Physical position number in the selected local bus: - Bus terminal modules: 0; - Local bus devices: 1 to 8
ID code:	Module identification code. It is printed as a *Module ID* on the modules in decimal form. (0 to 255$_{dec}$ corresponds to 00$_{hex}$ to FF$_{hex}$)

9.6 User Interface Messages

Command_Disabled_Confirmation009C$_{hex}$

Meaning: The previously issued command is disabled.

Syntax: The message consists only of one word, the message code (009C$_{hex}$). No further parameters follow.

Quit_Set_Parameter_Timeout_Confirmation00CF$_{hex}$

Meaning: The *Set_Parameter_Timeout_Constant_Request* (0034$_{hex}$) was executed.

Syntax: The message consists only of one word, the message code (0034$_{hex}$). No further parameters follow.

Quit_Enable_All_Messages_Confirmation00D6$_{hex}$

Meaning: The *Enable_All_Messages_Request* command (0047$_{hex}$) was executed.

Syntax: The message consists only of one word, the message code (00D6$_{hex}$). No further parameters follow.

Quit_Disable_All_Messages_Confirmation00D7$_{hex}$

Meaning: Das Kommando *Enable_All_Messages_Request* (0048$_{hex}$) was executed.

Syntax: The message consists only of one word, the message code (00D7$_{hex}$). No further parameters follow.

9.7 System Monitoring Messages

Software_Revision_Confirmation8087$_{hex}$

Meaning: Output of the controller board version. The parameter data is entered in the ASCII code.

Syntax:

Word 1	Code	
Word 2	Parameter count (n)	
Word 3	Parameter 1	Parameter block *Vendor name*
...	...	
Word 10	Parameter 8	
Word 11	Parameter 9	Parameter block *Host type*
...	...	
Word 26	Parameter 24	
Word 27	Parameter 25	Parameter block *Controller board*
...	...	
Word 42	Parameter 40	
Word 43	Parameter 41	Parameter block *Option*
...	...	
Word 58	Parameter 56	
Word 59	Parameter 57	Parameter block *Firmware*
...	...	
Word 65	Parameter 63	
Word 66	Parameter 64	Parameter block *Date*
...	...	
Word 79	Parameter 77	

Bit	15 8	7 0

Key:

Code:	Message code (here 8087$_{hex}$)
Parameter count:	Number of subsequent words (here 4D$_{hex}$ corresponds to 77$_{dec}$)
Vendor name:	Manufacturer of the controller board (16-byte ASCII string)
Host type:	Host PLC or computer system (32-byte ASCII string)
Controller board:	Controller board type (ASCII string of 32 bytes)
Option:	Further information on the controller board type (32-byte ASCII string)
Firmware:	Controller board firmware version (14-byte ASCII string)
Date:	Date of the firmware version (28-byte ASCII string)

9.8 Process Data Linkage Messages

Quit_Receive_Instructions_Confirmation00F2$_{hex}$

Meaning: The *Receive_Processing_Instructions_Request* command (005D$_{hex}$) was executed.

Syntax: The message consists only of one word, the message code (00F2$_{hex}$). No further parameters follow.

Processing_Instructions_Error_Confirmation80F3$_{hex}$

Meaning: The *Receive_Processing_Instructions_Request* command (005D$_{hex}$) could not be executed successfully.

Syntax:

Word 1	Code	
Word 2	Parameter count	
Word 3	00$_{hex}$	Error number
Word 4	00$_{hex}$	Parameter number

Bit	15 8	7 0

Key:

Code: Message code (here 80F3$_{hex}$)
Parameter count: Number of subsequent words (here 2)
Error number: The *error number* parameter specifies the type of the error (see subsequent table).
Parameter number: The *parameter number* parameter identifies the faulty parameter (00$_{hex}$ to FF$_{hex}$ corresponds to 0 to 255$_{dec}$) in the *Receive_Processing_Instructions_Request* command (005D$_{hex}$).

Table 9-3: Meaning of the *error number* parameter

Error number	Meaning
01$_{hex}$	All memory occupied. No further instructions can be accepted. The p*arameter number* parameter is in this case 0.
02$_{hex}$	Unknown instruction code
03$_{hex}$	Not enough parameters with one of the last instructions. In this case the p*arameter number* parameter is 0.
04$_{hex}$	No free internal RAM
05$_{hex}$	Reserved
06$_{hex}$	The logical output address does not exist.
07$_{hex}$	The logical input address does not exist.
08$_{hex}$	Invalid bit number. A bit number greater than 15 (for words) or greater than 7 (for bytes) was specified.

9.9 Event Processing Messages

Event_Error_Confirmation004C$_{hex}$

Meaning: An error was detected in the parameter check for programming the event. The error may occur with the following commands:
- Receive_Event_Request (002A$_{hex}$)
- Enable_Event_Number_Request (002B$_{hex}$)
- Disable_Event_Number_Request (002C$_{hex}$)
- Enable_All_Event_Request (002D$_{hex}$)
- Ensable_Event_Logical_Adress_Request (0036$_{hex}$)
- Disable_Event_Logical_Adress_Request (0037$_{hex}$)

Syntax: The message consists only of one word, the message code (004C$_{hex}$). No further parameters follow.

Quit_Receive_Events_Confirmation00BF$_{hex}$

Meaning: The *Receive_Events_Request* command (002A$_{hex}$) was executed.

Syntax: The message consists only of one word, the message code (00BF$_{hex}$). No further parameters follow.

Quit_Enable_Event_Number_Confirmation00DA$_{hex}$

Meaning: The *Enable_Event_Number_Request* command (002B$_{hex}$) was executed.

Syntax: The message consists only of one word, the message code (00DA$_{hex}$). No further parameters follow.

Quit_Disable_Event_Number_Confirmation00DB$_{hex}$

Meaning: The *Disable_Event_Number_Request* (002C$_{hex}$) command was executed.

Syntax: The message consists only of one word, the message code (00DB$_{hex}$). No further parameters follow.

Quit_Enable_All_Events_Confirmation00DC$_{hex}$

Meaning: The *Enable_All_Events_Request* (002D$_{hex}$) command was executed.

Syntax: The message consists only of one word, the message code (00DC$_{hex}$). No further parameters follow.

Quit_Disable_All_Events_Confirmation00DD$_{hex}$

Meaning: The *Disable_All_Events_Request* command (002E$_{hex}$) was executed.

Syntax: The message consists only of one word, the message code (00DD$_{hex}$). No further parameters follow.

Quit_Set_Event_Message_Type_Confirmation00DE$_{hex}$

Meaning: The *Set_Event_Message_Type_Request* command (004D$_{hex}$) was executed.

Syntax: The message consists only of one word, the message code ($00DE_{hex}$). No further parameters follow.

Set_Event_Message_Type_Failed_Confirmation$00DF_{hex}$

Meaning: The *Set_Event_Message_Type_Request* ($004D_{hex}$) could not be executed.

Syntax: The message consists only of one word, the message code ($00DF_{hex}$). No further parameters follow.

Quit_Enable_Event_Logical_Address_Confirmation$00E0_{hex}$

Meaning: The *Enable_Event_Logical_Address_Request* command (0036_{hex}) was executed.

Syntax: The message consists only of one word, the message code ($00E0_{hex}$). No further parameters follow.

Quit_Disable_Event_Logical_Address_Confirmation$00E1_{hex}$

Meaning: The *Disable_Event_Logical_Address_Request* command (0037_{hex}) was executed.

Syntax: The message consists only of one word, the message code ($00E1_{hex}$). No further parameters follow.

Event_Number_Indication8099$_{hex}$

Meaning: By means of bit-by-bit ORing of the event IDs, this message indicates all bits in a word, which occurred during a data cycle.

Prerequisites: Use only such event IDs for the 16 possible event definitions, which consist of a word where only one bit has been set:

0000 0000 0000 0001$_{bin}$ (0001$_{hex}$)
0000 0000 0000 0010$_{bin}$ (0002$_{hex}$)
0000 0000 0000 0100$_{bin}$ (0004$_{hex}$)
0000 0000 0000 1000$_{bin}$ (0008$_{hex}$)
0000 0000 0001 0000$_{bin}$ (0010$_{hex}$)
0000 0000 0010 0000$_{bin}$ (0020$_{hex}$)
0000 0000 0100 0000$_{bin}$ (0040$_{hex}$)
0000 0000 1000 0000$_{bin}$ (0080$_{hex}$)
0000 0001 0000 0000$_{bin}$ (0100$_{hex}$)
0000 0010 0000 0000$_{bin}$ (0200$_{hex}$)
0000 0100 0000 0000$_{bin}$ (0400$_{hex}$)
0000 1000 0000 0000$_{bin}$ (0800$_{hex}$)
0001 0000 0000 0000$_{bin}$ (1000$_{hex}$)
0010 0000 0000 0000$_{bin}$ (2000$_{hex}$)
0100 0000 0000 0000$_{bin}$ (4000$_{hex}$)
1000 0000 0000 0000$_{bin}$ (8000$_{hex}$)

Syntax:

Word 1	Code
Word 2	Parameter count
Word 3	All Events

Bit	15 0

Key:

Code:	Message code (here 8099$_{hex}$)
Parameter count:	Number of subsequent words (here 1)
All Events:	IDs of all events that occurred in a data cycle (bit-encoded). This word is generated by bit-by-bit ORing of the event IDs of all occurred events.

If the data of the time when the event occurred is required by the host by further event processing, evaluate an event message which supplies also the data word (Event_Data_Indication 809B$_{hex}$). Otherwise it is not ensured that the data is of the same time as the event.

Event_Data_Indication809B$_{hex}$

Meaning: This message indicates all events that occurred during a data cycle. The event ID and the data word of the input module where the event was initiated are transferred as parameters. The event ID must habe been defined in advance.

Syntax:

Word 1	Code	
Word 2	Parameter count (n)	
Word 3	Event ID	for 1st event
Word 4	Data word	
	...	
	Event ID	for xth event
Word n+2	Data word	

Bit	15 0

Key:

Code:	Message code (here 809B$_{hex}$)
Parameter count:	Number of subsequent words (here number of events multiplied by 2)
Event ID:	IDs of the events that occurred in a data cycle (bit-encoded).
Data word:	Data word of the input module where the event specified in the line before was initiated.

Quit_Read_Event_Counter_Confirmation811D$_{hex}$

Meaning: This message is the positive or negative acknowledgment of the *Read_Event_Counter_Request* command (0114$_{hex}$).

Positive acknowledgment: This message transfers the counts of the requested event counters as positive acknowledgments. With the positive acknowledgment, the *Result* parameter is 00$_{hex}$.

Syntax:

Word 1	Code	
Word 2	Parameter count (n)	
Word 3	Result	ID code
Word 4	Event count	
Word 4	Event count	
	...	
Word n+2	Event count	

Bit	15 8	7 0

Key:

Code:	Message code (here 80F4$_{hex}$)

Parameter count:	Number of subsequent words (here number of read-out event counters, max. 16_{dec}).
Event count:	Counts of the read-out event counters. The order of ouputs corresponds the the order specified with the request (*Read_Event_Counter_Request* (0114_{hex}) command). If you had entered a 0 there to read read out all counters, the order of outputs corresponds to the order specified in the event definition (*Receive_Events_Request* ($0002A_{hex}$) command).

Negative acknowledgment: This message does not transfer counts but error codes as negative acknowledgments. With the negative acknowledgment the *Result* parameter is FF_{hex}.

Syntax:

Word 1	Code	
Word 2	Parameter count (n)	
Word 3	Result	Error class
Word 4	Error code	Additional code
Word 5	Additional code	00_{hex}

Bit	15 8	7 0

Key:

Code:	Message code (here $811D_{hex}$)
Parameter count:	Number of subsequent words (here 3).
Result:	For the negative acknowledgment FF_{hex}.
Error class:	Reserved, (00_{hex}), currently with no meaning
Error code:	Reserved (00_{hex}), currently with no meaning
Additional code:	Event ID where an error was detected.

Appendix A

Technical Appendix

A Technical Appendix

A.1 Technical Data of the Controller Boards

General data

Permissible temperature range	
- Operation (inlet air temperature):	0^{o}C to +55^{o}C
- Storage (inlet air temperature):	-20^{o}C to +70^{o}C
Permissible humidity	
- Operation:	75% (non-condensing)
- Storage:	95% (non-condensing)
Insulation strength:	0.5 kV

Mechanical design

Format:	Long standard PC AT board
Front plate:	PC card holder
Dimensions:	338.5 mm * 114.3 mm
Ground	
- IBS PC CB/I-T:	Approx. 340 g
- IBS PC CB/COP/I-T:	Approx. 500 g
- IBS PC CB/RTX486/I-T:	Approx. 500 g

Host interface

AT Bus interface	
- Connector:	62- and 36- pos. edge connector
Address requirements per controller board (see Chapter 4)	
- in the host memory:	4 Kbytes
- in the I/O area:	8 bytes
- Interrupts:	One
Permissible number of controller boards:	Up to four per host

Supply

Permissible voltage (including ripple):	+5 V DC ± 5 %
Permissible ripple:	100 mV_{pp}
Current consumption	
- IBS PC CB/I-T:	typically 1.2 A
- IBS PC CB/COP/I-T:	typically 1.5 A
- IBS PC CB/RTX486/I-T:	typically 1.5 A
Battery:	6 V, 1350 mAh e.g: Varta 2/CR 2/3 AA

InterBus-S

IBS master board:	MA5 board
Remote bus connection	
- Interface:	2-wire remote bus
- Connector:	SUB-D9, female
- Transmission rate:	500 Kbits/s
- Max. number of IBS devices:	320
- Max. number of remote bus devices:	256
- Max. number of bus segments:	256
- Max. number of process data words:	256
- Max. number of PCP devices:	256
Diagnostic and parameterization interface	
- Protocol:	RS-232C-compatible
- Connector:	Subminiature D9, male connector

***COP 386* (IBS PC CB/COP/I-T) coprocessor board**

Operating system:	TDOS
Dimensions:	78.74 x 113.03 mm
Processor:	80386 SX
Clock frequency:	25 MHz
Chipset:	Chips & Technologies 82C836
Main memory:	2 Mbyte DRAM 128 kbyte CMOS-RAM (battery-backed-up)
EPROM 1:	128 kbytes
EPROM 2:	Not fitted
Power supply:	+5 V ±5%; typ. 380 mA
Serial interface:	COM1, 9-pos.
Interface chip:	UART NS 16C450

***COP 486* (IBS PC CB/RTX486/I-T) coprocessor board**

Operating system:	RTXDOS
Dimensions:	78,74 x 113,03 mm
Processor:	486 SXLC-40, double clock, 8 KByte Cache
Clock frequency:	40 MHz internal 20 MHz external
Chipset:	Chips & Technologies 82C836
Main memory:	2 Mbyte DRAM, 128 kbyte CMOS-RAM (battery-backed-up)
Flash EPROM 1:	256 kbytes
Flash EPROM 2:	256 kbytes
Power supply:	+5 V ±5%; typ. 380 mA
Serial interface:	COM1 with FIFO, 9-pos.
Interface chip:	ST16C550JW-44

Appendix B

Document Index

B Document Appendix

B.1 Figures

Section 1

Section 2

Section 3

Section 4

Section 5

Section 6

Section 7

Section 8

B.2 Tables

Section 1

Section 2

Section 3

Section 4

Section 5

Section 6

Section 7

Section 8

Section 9

B.3 Index

L

M

N

O

P

R

S

T

V

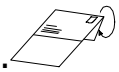
W

We Are Interested in Your Opinion!

We would like to hear your suggestions, wishes and criticisms concerning this InterBus manual. Therefore, we would appreciate your answers to the questionnaire overleaf. No matter how small your contribution, we will deal with any hint or comment.

This reply can be sent as a fax, in an envelope, or directly, after folding (1. / 2.) and sticking it together (with adhesive tape).

Thanking you for your cooperation !

1. 

Phoenix Contact GmbH & Co.
Produktmarketing InterBus/ME-DOK
Flachsmarktstraße 8 - 28
32825 Blomberg
Germany

2. 

FAX-NO.: *49-(0)5235-331199

FAX Reply
Phoenix Contact GmbH & Co.
Produktmarketing InterBus/ME-DOK

Date: .....

No. of pages: .....

Sender:

Company: ..... Contact: .....

..... Dept.: .....

Address: ..... Title: .....

City, ZIP code:..... Telephone: ..... / .....

Country: ..... Fax: ..... / .....

Manual data:

Type: ..... Revision: ..... Order No: .....

My Opinion on the Manual

Form

	Yes	In part	No
Is the Table of Contents clearly arranged?	❒	❒	❒
Are the figures/diagrams easy to understand/meaningful?	❒	❒	❒
Does the page layout make it easy to locate the required information?	❒	❒	❒
Is the chapter overview detailed enough?	❒	❒	❒
Do the corresponding explanations of the figures suffice?	❒	❒	❒
Does the quality of the figures (clear/self-explaining) meet your expectations/requirements?	❒	❒	❒
Do the index entries guide you to the relevant information?	❒	❒	❒

Contents

	Yes	In part	No
Are the technical terms easy to understand/meaningful?	❒	❒	❒
Is the wording always easy to understand/meaningful?	❒	❒	❒
Are the index entries easy to understand/meaningful?	❒	❒	❒
Which reference list (Table of Contents, Figures, Tables, Index) do you use most when you search for information ?			
Is any important information missing? If so, which?	❒	❒	❒
Are the examples practice-oriented?	❒	❒	❒
Is the manual easy to handle?	❒	❒	❒
Do you wish more space for your own notes?	❒	❒	❒

I have the following suggestion(s)/comment(s):

.....
.....
.....
.....
.....
.....

DNR 2533

5050CF17